Toni Liski

3-D collision checking for improving machine operator's spatial awareness

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 29.9.2014

Thesis supervisor:

Prof. Ville Kyrki

Thesis advisor:

D.Sc. (Tech.) Sami Terho

Aalto University
School of Electrical
Engineering

AALTO UNIVERSITY
SCHOOL OF ELECTRICAL ENGINEERING

ABSTRACT OF THE
MASTER'S THESIS

Author: Toni Liski

Title: 3-D collision checking for improving machine operator's spatial awareness

Date: 29.9.2014 Language: English Number of pages: 8+61

Department of Electrical Engineering and Automation

Professorship: Automation Technology Code: AS-84

Supervisor: Prof. Ville Kyrki

Advisor: D.Sc. (Tech.) Sami Terho

In this thesis, methods to improve machine operator's spatial awareness are studied. The aim is to allow a machine to be teleoperated from distance. As a background, the research on the methods for environment modelling is reviewed, the comparison of environment representation methods, and an introduction to collision checking techniques is made. In this work, a novel approach to represent partly dynamic environment obtained from a staticly mounted range sensor was developed. In addition, a 2-D collision checking system using this representation to warn of possibly incoming collisions was developed. The developed environment model works precisely with partly dynamic environment when incoming range data is accurate enough and does not contain major artifacts or noise. When the environment model is created successfully, the collision checking system works accurately and is fast compared to pure 3-D collision methods.

Keywords: Point cloud, collision avoidance, 3-D world, safety

AALTO-YLIOPISTO
SÄHKÖTEKNIIKAN KORKEAKOULU

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä: Toni Liski

Työn nimi: 3-D törmäystarkastelu työkoneen käyttäjän ympäristötietoisuuden parantamiseksi

Päivämäärä: 29.9.2014 Kieli: Englanti Sivumäärä: 8+61

Sähkötekniikan ja automaation laitos

Professuuri: Automaatiotekniikka Koodi: AS-84

Valvoja: Prof. Ville Kyrki

Ohjaaja: TkT Sami Terho

Tässä työssä tutkittiin menetelmiä työkoneen käyttäjän ympäristötietoisuuden parantamiseksi siten, että työkonetta olisi mahdollista etäohjata. Työssä tutkitaan menetelmiä, joilla ympäristöä voidaan mallintaa tehokkaasti, ja miten luotua ympäristömallia voidaan hyödyntää törmäystarkastelusovelluksessa. Työssä päädyttiin toteuttamaan uudenlainen menetelmä, jolla voidaan yksinkertaisesti esittää osittain muuttuva ympäristö kiinteästi sijoitetulla kameralla kuvattuna. Lisäksi toteutettiin törmäystarkastelu ja yksinkertainen mahdollisesta törmäyksestä varoittava algoritmi hyödyntämällä työssä luotua ympäristömallia. Luotu ympäristömalli toimii hyvin ja sietää liikettä ympäristössä, kunhan saatava ympäristömittaustieto on riittävän tarkkaa eikä sisällä merkittäviä stereokuvauksesta johtuvia virheitä. Kun ympäristömalli on luotu onnistuneesti, kehitetty törmäystarkastelu toimii tarkasti ja on erittäin tehokas verrattuna kolmiulotteisiin menetelmiin.

Avainsanat: Pistepilvi, törmäystarkastelu, 3-D avaruus, työturvallisuus

Preface

I want to thank my advisor D.Sc. Sami Terho for the invaluable help with the technical issues I encountered during the project and the assistance I warmly accepted regarding the structure and content of this thesis. I want to thank Konsta Hölttä and Erkka Mutanen for a spell-checking of the thesis and the peer support during the project. In addition, I want to thank my supervisor Prof. Ville Kyrki for the valuable support and the pieces of advice I have received.

The research was a part of FAMOUS project of the Energy and Life Cycle Cost Efficient Machines (EFFIMA) research program, managed by the Finnish Metals and Engineering Competence Cluster (FIMECC), and funded by the Finnish Funding Agency for Technology and Innovation (TEKES), several companies, and research institutes.

Otaniemi, September 29, 2014

Toni Liski

Contents

Symbols and Abbreviations

Symbols

a	cell of model
$\mathbf{b}$	mean of $\mathbf{x}_i$
d	disparity
$\hat{d}(a)$	measured distance in a
$d(z_{k,a})$	distance in cell a in model z_k
f	focal length
h	height of environment
i, j, r, s, k	indices
m_i	vector of bounding box corner
n	leaf node or number of data points
n_i	normal vector of the side s_i
p_e	point on the environment
p_r	point on the search ray
$p_{i,k}$, $p_{i,k,l}$	point around the bounding box
$p(x)$	probability of x
$p(x, y, z)$	point on image plane with a value z
$\dot{p}_r$	r'th point on the side of the bounding box
s_i	side vector of bounding box
t	time of measurement
u_i	direction vector of search ray
$\hat{u}_i$	unit direction vector
w	width of environment
x	x-coordinate of 2-D point
$\mathbf{x}_i$	2-D NDT point
y	y-coordinate of 2-D point
z	sensor measurement or depth
z_k	current environment model
z_{k+1}	new environment model
z_{max}	maximum depth value
B	baseline of stereo cameras
I	inverval value
L	length of the search ray
$L(n\|z)$	log-likelihood of $P(n\|z)$
M	scaling matrix
$N(q, \Sigma)$	normal distribution
$P(X, Y, Z)$	3-D point
$P(n\|z)$	probability of n to be occupied having z
$P(z_{k,a})$	probability of cell a in model z_k
P_{prior}	prior probability
P_{post}	posterior probability
P_{limit}	threshold probability
R	resolution
$\mathbf{S}$	size matrix
S_v	scaling matrix
T	transformation matrix
T_v	translation matrix
Σ	covariance matrix

Abbreviations

$2\frac{1}{2}$-D	Range image or depth map type representation
AABB	Axis-Aligned Bounding Box
BV	Bounding Volume
BVH	Bounding Volume Hierarchy
CMT	Consensus-based Matching and Tracking of Keypoints
DOP	Discrete Orientation Polytopes
FDH	Fixed Directions Hull
KLT	Kanade-Lucas-Tomasi tracking algorithm
LADAR	LAser raDAR
LIDAR	LIght Detection And Ranging
LRF	Laser Range Finder
NDT	Normal Distribution Transformation
NDT-OM	Normal Distribution Transformation – Occupancy Map
OBB	Oriented Bounding Box
ODE	Open Dynamics Engine
OM	Occupancy Map
PCL	Point Cloud Library
ROS	Robot Operating System
STOC	STereo On a Chip
TLD	Tracking-Learning-Detection, a tracking algorithm
ToF	Time-of-Flight

1 Introduction

Vehicle teleoperation means simply: *operating a vehicle at a distance.* Teleoperation has been common in mining industry for a while, mainly for hard working conditions and occupational safety. Commercial companies have proven that teleoperation and partly autonomous operation both increase comfort and also reduce maintenance costs.

By developing techniques needed for safety related teleoperation, overall costs for teleoperating vehicles have been reduced significantly. Currently on-going studies concentrating on teleoperation have been expanded to already unpleasant working areas, such as isolated tasks. One this kind of a task is driving a forestry machine. The driver might need to travel long distances daily just to arrive in the middle of the forest and start to operate the machine, alone. In the forest there is the safety aspect, as in the case of an accident, nobody will notice the accident until it might be too late. Another big aspect is the cost. It might take hours just to travel to the current logging site, when the operator could have worked considerably longer days by teleoperating a machine from the distance. One driver could also use multiple semi-autonomous machines by teleoperating those from a single location.

One disadvantage in teleoperation is the lack of spatial awareness. Without the ability to sense depth on the screen, it is undoubtedly hard to avoid the collisions with surrounding fixed objects that could have been avoided when the operator is on the spot. Studies have proven that expensive 3-D monitoring and stereo vision systems are unpleasant to use and do not guarantee the same accuracy as operating locally. Some other methods are needed.

In this thesis the methods to avoid collisions between a manipulated load and environment are studied. The starting point of the study is to build a system to warn the operator of the possible oncoming collisions. The research questions of this thesis can be phrased as:

> How can a machine operator's spatial awareness be extended with collision checking techniques that utilize environment data from 3-D range sensor and information of load geometry?

For the reason this thesis is the part of the wider totality of the spatial awareness extending project, it will be necessary to include background from the whole project into this research. The background and the use case are reasonable large area of study, so experimental part of this thesis concerns only part of it. In addition, there are already theses written on the other parts [1, 2].

The experimental collision checking case can be divided roughly into five parts:

1. sensing the environment by a stereo camera
2. constructing a point cloud from the stereo images
3. detecting dynamic objects
4. creating the environment model
5. detecting possibly oncoming collisions

This thesis concentrates on the last two subjects of the case, but also gives a short overview of the other three subjects. Although all the methods this thesis presents are possible to generalize into many fields of research, the experiment part of the thesis concentrates mainly on extending user's spatial awareness when teleoperating a machine with a crane or a boom for lifting objects.

In the first section, the background and related works in this topic are studied. Background section contains an overview of multiple range sensing techniques, environment representation formats, and background of collision detection techniques. Also optical flow tracking and three algorithms based on it are introduced. In the end of the section, three software libraries for robotics applications are introduced. In section three, a new environment representation method and a collision checking algorithm based on that representation format are developed. The section also contains overview of the user interface implemented for the testing the developed algorithms. In section four the behaviour of the developed algorithms are studied. The aim of the experiments is to prove the correctness and accuracy of the algorithms. The results are captured partly from the real testing environment and partly from the developed testing applications. Finally, conclusions and future work are discussed.

2 Background

This section contains an overview of multiple range sensing techniques, environment representation formats, and background of collision detection techniques. The range sensing is a vital part to create an accurate model of the environment. An environment representation format has to be selected according to a case. Also multiple collision checking techniques exist, so the background of those are introduced here. Optical flow tracking and an algorithm based on it will be used to find 3-D coordinates of the end-effector by combining range data and a single video frame. In the end of this section, three software libraries for implementing the collision detection system successfully are introduced.

2.1 Environment representation

Environment awareness is one of the most central functions of robots. Some representation is necessary for path planning and intelligent movements of the robot. For wheel based robots it is often sufficient to have 2-dimensional map-based representations of the road network or corridors inside the building. If a robot is able to operate in multiple dimensions, it is often recommended to have 3-D information of the environment. Many formats to store 3-D data are available and they are suitable for different purposes. For some task a 2-D representation is adequate, but purely 2-D techniques are not discussed in this work. For 3-D representation, although there are many techniques available, representations as point clouds and octrees are the most commonly used in robotics applications [3, pp. 530-532]. Many range sensing techniques produce just a set of the points, but they are not necessarily the most efficient way to encode the environment.

Desired features of 3-D representation techniques normally include easy and efficient updatability, ability to differentiate between free and unobserved space, and low memory usage. Depending on a purpose, also collision checking might be desired feature. Line and plane based approaches are often practical for representing man-made structures and are efficient especially for indoor environments, but are not suitable enough to encode, for instance, trees or terrain.

Range data is a $2\frac{1}{2}$ or 3-dimensional representation of the scene around an observer. The pure 3-D aspect arises when sensors capable of recording (X, Y, Z) information of each point are present. Often only a single range image is used, so only one side of an object can be detected at once. This leads to the name *two-and-a-half*-dimensional image. It is also called *range image* format, where data is

encoded as a function $d(i, j)$, which records the distance d to the the corresponding scene point (X, Y, Z) for each pixel (i, j). [3, p. 521]

Point clouds are collections of 3-D points where each point represents corresponding spot on a wall, ground, or on any other structure in 3-D world. The point cloud is the easiest 3-D structure to build based upon, for example, laser range data or other range sensor. In that case, points are uniformly distributed over the whole scene. The points fetched from a stereo vision system usually correspond to some detected features, such as corners, which leads that they are not uniformly distributed any more.

The representation type of point cloud depends on the use case. Point clouds can be represented as an unorganized list of points, but one disadvantage of this format is that the whole list must be iterated over in order to do anything with it. Another representation format is so called *organized array* where points are organized in a two-dimensional array. Each cell in the array contains a single point where X and Y-indices of the cell in array correspond to X and Y-coordinates of the point at the environment. This format is quite similar to a *range image*. The disadvantage of this format is then inability to represent objects successively located, that is, obstacles behind a wall. However, much more sophisticated methods to represent 3-D points have been developed. One of those is an occupancy grid map.

The *occupancy grid map* [4] is a popular and effective approach to present the environment. It is based on posterior probability that the corresponding area in the environment is occupied by an obstacle. The advantage of the occupancy grid map is that it can be updated when new data comes available, and it allows fully dynamic environments. Additionally, it offers constant time access to every grid cell and provides ability to present empty (free) and unknown (unobserved) areas separately. The naïve implementation of occupancy grid map suffers from high memory usage. [3, p. 855]

One of the most recent implementations of occupancy grid map is *OctoMap*[5]. The implementation is widely used and it is included also in the Robot Operating System (discussed later). The main property of presented approach is that the algorithm allows efficient and probabilistic updates of occupied and free space while keeping the memory consumption low.

OctoMap is based on octrees. The octree is a hierarchical data structure for spatial subdivision in 3-D. Each node in an octree, called *voxel* (volume pixel), represents the space contained in a cubic volume. The volume can be recursively subdivided into eight sub-volumes until a given minimum voxel size or depth is

reached. Advantage of this hierarchical tree is that it can be cut at any level to obtain specific resolution and faster processing times (see Figure 1). [5]

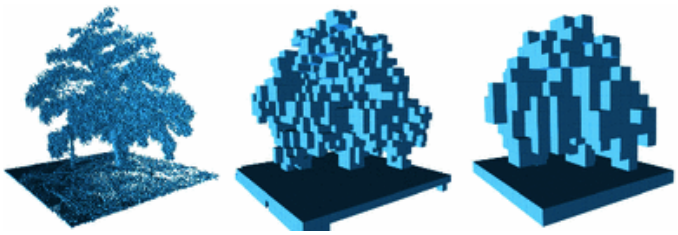

Figure 1: Multiple resolutions of the same octomap. Occupied voxels are displayed in resolutions 0.08, 0.64, and 1.28 m (image source [5]).

In the terms of sensor fusion, the octomap takes advantage of the technique introduced by Movarec and Elfes[4]. In Equation (1) the probability $P(n|z_{t:t})$ of a leaf node n to be occupied given sensor measurements $z_{1:t}$ is estimated by

$$P(n|z_{1:t}) = [1 + \frac{1 - P(n|z_t)}{P(n|z_t)} \frac{1 - P(n|z_{1:t-1})}{P(n|z_{1:t-1})} \frac{P(n)}{1 - P(n)}]^{-1} \tag{1}$$

where z_t is the current measurement, $P(n)$ is a prior probability of the node being occupied, and $P(n|z_{1:t-1})$ the previous estimate. The term $P(n|z_t)$ denotes the probability of voxel n to be occupied given measurement z_t. The common assumption of a uniform prior probability leads to $P(n) = 0.5$ and by using the log-odds notation, Equation (1) can be rewritten as

$$L(n|z_{1:t}) = L(n|z_{1:t-1}) + L(n|z_t), \tag{2}$$

with

$$L(n) = log[\frac{P(n)}{1 - P(n)}]. \tag{3}$$

In words, L is the log-likelihood of a volume n being occupied, and z_t is the sensor measurement at time t.

Triangulated surfaces describe an object or a scene by a set of triangular patches. More general polygonal surface patches have also been used, but triangles are the most common because they are simpler and PC graphics accelerator cards use triangles to represent 3-D environments in games. The size of triangles varies between different objects. Planes can be represented using few triangles, while more complex objects need several (hundreds, thousands) triangles. The triangulated surface might be complete in a sense that all object surfaces are represented by triangles,

or then there might be disconnected surface areas and internal holes. In order to use triangulated surfaces to navigation or grasping, it is recommend that the representation should enclose the represented objects entirely. [3, p. 531]

Elevation map, or *height-map*, is a representation technique constructed from range images. It stores $2\frac{1}{2}$-dimensional information of the environment. Elevation map suffers from ability to represent the overhanging environmental objects like bridges. For example, a robot that uses elevation maps cannot plan a path at the same time under and over a bridge. Effort has been made to improve elevation maps, and one approach containing extended functions is the *multi-level surface map* [6]. That method uses multiple elevation clusters, each with its own mean and variance. However this method neither store the knowledge between free and unobserved area.

The *Normal Distribution Transform* (NDT) models the probability of a point at a certain position by a collection of normal distributions [7]. Similar to the occupancy grid, the space around the observer (robot) is subdivided regularly into cells with a constant size. Then for each cell containing at least three points, the following is done [7]:

1. Collect all 2-D points $\mathbf{x}_{i=1..n}$ in this area
2. Calculate the mean $\mathbf{b} = \frac{1}{n}\sum_i \mathbf{x}_i$
3. Calculate the covariance matrix $\Sigma = \frac{1}{n-1}\sum_i (\mathbf{x}_i - \mathbf{q})(\mathbf{x}_i - \mathbf{q})^t$.

The probability of measuring a sample at 2-D point x contained in this cell can now be modeled by the distribution $N(q, \Sigma)$:

$$p(x) = exp(-\frac{(\mathbf{x}_i - \mathbf{q})^t \Sigma^{-1} (\mathbf{x}_i - \mathbf{q})}{2}). \tag{4}$$

The difference between occupancy grid and NDT is that where occupancy grid represents the probability of a cell being occupied, NDT represents the probability of measuring a sample for each position within the cell. In [8] the 3-D NDT was demonstrated to outperform the occupancy maps in terms of representing the observations. The paper also shows that NDT is less sensitive to the choice of the grid size. NDT and Occupancy Map can also be combined. *Normal Distributions Transform Occupancy Map*[9] (later NDT OM) is a novel approach designed for real time 3-D mapping in a large-scale, dynamic environment. Where the NDT update assumes a static environment, OM part enables the estimation for probabilistic occupancy for all cells. The NDT-OM is capable of modeling free space, like occupancy map, while it can be updated real-time, even for large environments.

2.2 Sensing the environment

Sensing the environment in this thesis means simply the usage of a range sensor. An example of a one-dimensional range sensor is a proximity sensor, for instance a laser range finder, that uses a laser beam to determine the distance to an object. In 3-D robotics, a single distance is not enough, so more advanced solutions to construct 3-D range images or other 3-D environmental representations are needed. Often the word *range* means particularly capture of the three-dimensional structure of the world, from the viewpoint of the sensor [3, p. 521]. In practise, the range is the depth measure to the nearest surfaces. In this context only 3-dimensional ranging sensors are discussed.

The sensing can be divided into two different categories: active and passive sensing. The passive approach means that the sensing method does not use a source of energy to illuminate the scene [10]. The advantages of passive methods are cost, simplicity of imaging hardware, compatibility with human visual process. Challenges arise from the loss of information associated with the perspective mapping of a 3-D scene onto a 2-D image. The active approaches for 3-D sensing use special illumination sources and sensors. Most state-of-the-art methods use laser radar, or structured lighting patterns. [3, p. 90]

In the following sub-sections different sensing methods are discussed and slightly compared. First some laser-based ranging sensors are presented and after that some image based system are introduced.

2.2.1 Time-of-Flight cameras

One attempt to sense surrounding 3-D world has been successfully done by *time-of-flight* (ToF) range camera [11]. The technique in ToF camera is based on modulated light reflected from the objects in the view. Every pixel on a two-dimensional image sensor samples the amount of modulated light reflected from the object. The distance from the object is then determined from the phase-shift between the emitted and returning light signal [12]. ToF system contains no moving parts, which is the most significant benefit compared to other 3-D range sensing sensors [13]. Another is admittedly a high frame rate that goes up to 50 frames per second (fps) [14, 11].

However, there still exist many fundamental problems [15] with ToF. The first drawback is the *wrap-around* problem, which occurs when measuring objects on a deep distance range. If, for instance, the phase 2π corresponds the distance of 8 meters, then two distances $0,2m$ and $8,2m$ lead to the same range value. In

theory, there is a possibility to use *Amplitude Threshold* parameter to compensate this problem, but it brings out a new question – how to find suited parameter for a changing real world environment. [15]

Another significant drawback of the established ToF sensors is extremely narrow field of view, with a limited standard view of $47° \times 39°$ [15]. In addition, systematic errors related to *integration-time*, *built-in pixels*, *amplitude*, and *temperature* exist [11].

3-D ToF sensors are sometimes called *flash LiDAR* or *flash LADAR*, which are actually acronyms. [3, p. 528]

2.2.2 Laser Range Finders

Laser range finders (LRF) are based upon an established technique similar to ToF sensors. There is a single laser pulse and one sensor that measures the time between emitting and receiving the reflected light signal. In order to get 2-D information about environment, there is a need for an actuator that causes laser pulse to *spin around* [16]. These kind of 2-D scanners are nowadays small, affordable, and their power consumption is low enough for them to be used on a mobile robot. One well-known 2-D LRF is introduced in [16].

The recent studies in the field of the robotics require precise 3-D information of environment. The 3-D model is used to plan safe trajectories for mobile robots (arms) as well as to detect and avoid obstacles. As 3-D laser range finders (like Velodyne [17]) are still expensive and too massive for many mobile robots, other attempts, like actuated laser range finders (aLRF), still prevail. An actuated laser range finder is normally constructed by mounting a normal 2-D LRF on the top of a tilt-type servo actuator. The actuation could happen along any of roll, pitch, or yaw axes. Adding third dimension to sensing increases value of laser scanning substantially, but at the expense of speed and accuracy. [18]

The technique combining ToF range cameras and actuated aLRF is called *three-dimensional laser range finder* (3-D LRF), *light detection and ranging* (LIDAR), or *laser radar* (LADAR). 3-D LRF is currently a state-of-the-art product, but it still suffers from its pretty massive size and high price. Recent sensors, like Velodyne HDL-64E [17], provides however reasonable frame rate from 5 Hz to 15 Hz and full 360° azimuth and 26.5 degree elevation field of view. The drawback is that price is rather high, around $75,000, due the 64 fixed-mounted lasers in the sensor.

2.2.3 Structured light cameras

Structured light is an active range sensing technique, that means, it is a triangulation method with a known illumination source texture [19]. The technique behind structured light is well established [10]. Compared to multiple cameras in stereo vision, the implementation of a structured light system uses just one camera and a lighting source that emits a predefined or otherwise well known pattern on the target. The pattern can be almost anything – first implementations used a single dot or line, later multiple lines, circles, cross-hairs, thick stripes, coded binary patterns, color-coded-stripes [20], and random textures have been used [21, 22]. The pattern on the object is then detected by a camera, and used to compute a structure of the object or a range image of the view [23, 24].

The lighting source normally emits visible light, but not necessarily. One successful implementation using infra-red light is Kinect sensor designed for consumer market [25]. Kinect sensor was released by Microsoft in November 2010 as an accessory for a game console. Many researchers have found it an interesting alternative for mobile robotics, especially due its low price and high frame rate. The Kinect sensor is a compact solution including both infra-red and RGB cameras, which can be used to construct accurate and coloured 3-D depth images. In addition to two cameras, the sensor contains an infrared laser, that emits a constant pattern of speckles projected onto the scene. The functionality is based upon an active triangulation process where the emitted pseudo-random pattern is detected using infra-red camera. The 3-D model is constructed from the depth-disparity relation of infra-red camera and laser projector, while RGB camera is used just to colour the constructed point cloud. [26]

The Kinect sensor is suitable especially for indoor environments where distances are limited [18], as the limited power of the projector can cause inaccuracy at long distances. The frame rate of about 30 fps, and colored point clouds with about 300 000 points in every frame enables Kinect to be used to create a complete point cloud of indoor environment even in real time [26]. The Kinect sensor does not depend on the object texture or require additional lightning, so it is suitable to detect featureless materials like walls or other man-made objects in a dark room.

2.2.4 Stereo vision

Stereo vision is one of the oldest research topics in the history of Computer Vision. Previously its use in robotics was limited by the large amount of computing required

for the correlation matching of each pixel. The rapid development on the processor techniques has speeded up the implementation and use of stereo systems [19]. Today, the real-time implementation of a stereo system is possible using normal PC hardware.

A stereo vision system consists of at least two RGB or greyscale cameras [19]. The RGB information is not necessarily needed, but it can improve results of feature matching as well as accuracy of the depth image. Stereo cameras can be used to create a range image of a scene by recognizing common features across the images [18]. Resolving the triangulation of these correspondences with a known baseline produces a depth value for each pixel observed.

The design of a stereo vision system has three degrees of freedom. One of the degrees is the baseline of (the distance between) the cameras. As it increases the disparity become larger, making it possible to estimate depth to greater precision. The disadvantage is though the occlusion and larger computation time, so the disparity search range needs to be set carefully. If optical axes of a stereo pair are not aligned, the images captured must be rectified in order to process they effectively. [27, p. 441]

Range sensors usually produce range data in a format that it is rather easy to construct a point cloud based upon it. Stereo cameras, however, produce only a pair of images without actual depth information. The depth image must then be constructed using the knowledge of extrinsic and intrinsic parameters of the cameras [27, p. 381]. When the physical measurements are known, the remaining part is to match points in both images. This is usually done by looking for corresponding features in parallel images. The depth z can then be calculated from disparity d between corresponding points with a formula

$$d = Bf\frac{1}{z} \tag{5}$$

where B and f are baseline and focal length of the cameras [28]. The way, how the points to match are selected, varies considerably and is out of the scope of this work. This and other basic methods to solve stereo correspondence problem are introduced in [29], while several more advanced methods are compared in [30, 31, 32].

Nowadays cameras are usually connected to a computer that computes range image from the parallel images. There have also been alternatives, like Videre STOC (**ST**ereo **O**n a **C**hip)[15] – as the name tells, the sensor includes the chip capable of stereo image processing – but the company seems to have vanished later, probably due the high cost of the products compared to cheap PC hardware nowadays.

2.3 Collision detection and avoidance

Safety issues in robotics primarily contain a concept of collision avoidance. It is an essential feature of a robot, especially when operating in unstructured environment or when a robot share its workspace with humans [33]. Accidental collisions will harm both robot and target, so they should be avoided. Robots and manipulators should always be equipped with a reactive collision detection, but it should not be the primary way to avoid collisions, as collision has already occurred when a reactive, for example torque-based, sensor is activated. Typically robots are equipped with multiple sensors that can be used to detect collisions [34]. In 2-D environment, a distance sensor or a few is understandably enough to avoid collisions on the course of a mobile robot, but in terms of robot arms and manipulators with several degrees-of-freedom, they are not sufficient any more.

2.3.1 Concept of collision checking

It is important to make a difference between *collision detection* and *collision avoidance*, although the terms are often mixed. The collision avoidance, sometimes called *collision prediction*, is technique to avoid collisions in advance. Collision avoidance methods use the knowledge of the environment to plan safe routes or trajectories for the robot. The planning is a complex task for even a simple robot arm with a couple of joints. In dynamic environments, collision avoidance methods can be used only partly and safe trajectories must be recalculated if the environment model changes [35].

In [36] collision avoidance approaches are divided into two categories: *global* and *local*. The global approach means path planning and it is well-suitable for static environments. As mentioned before, planning is a complex and computationally expensive task, so for dynamic environments it is too expensive to be done repeatedly. The local approach uses only a small fraction of the environment model to generate robot control. The disadvantage is that the local methods cannot produce optimal solutions and are easily trapped in local minima such as U-shaped rooms. The *local method* is very similar to the *collision detection*, so they both are used for reactive controls. The key advantage of local techniques over global ones lies in their low computational complexity. For mobile robotics, many local collision avoidance methods have been developed, such as the collision detection with enlarged objects, nearness diagrams, potential field methods, vector field histogram, and a dynamic field method.

The main principle of collision detection is the intersection testing between two known, usually geometric, models. If the models are intersecting or about to intersect, then the collision of the corresponding objects is about to occur. The geometric model may be a polygonal object, a spline, or an algebraic surface [37]. In order to apply collision detection techniques to dynamic or non-structured environments, other representation methods also exist. The point clouds[38], voxel maps[39], or octrees[40] are easy and efficient to update and use to detect the collisions.

To make things more complicated, the concept of collision detection can still be divided into two parts. *Static collision detection* involves detecting intersection between two stationary objects, at discrete points in time, during their motion. At each point in time the objects are treated as if their movements were halted and they had zero velocities. In contrast, *continuous collision detection* (also called *dynamic collision detection*) considers the full motion including the velocities and directions. The dynamic collision detection is time-based and therefore can usually report the exact time of an expected collision and the points of the first contact. In order to take advantage of a static checking successfully, intervals between two successive tests must be small enough. Static tests are however popular, because they are much cheaper than dynamic checking. [37][41, pp. 16-17]

In a dynamic environment, any object can potentially collide with any other object. It means that, in the worst case, n objects require $O(n^2)$ pairwise tests to detect collisions [41, p. 14]. Due to the quadratic time complexity, it is easy to see that the naïve testing of an every object pair for collision quickly becomes expensive. To speed up the process, the collision handling of multiple objects can be separated into two phases: *broad phase* and the *narrow phase*[42].

2.3.2 Broad and narrow phase

Hubbart [42] was first who classified collision detection in terms of a broad and a narrow phase. The idea of the phases is to reduce computational load by performing a coarse test in order to prune unnecessary pairwise tests. The broad phase collision detection identifies disjoint groups of possible intersecting groups (see fig. 2). The broad phase determines objects which should be tested during the narrow phase collision detection. Usually the broad phase approximates the objects with boxes or cubes in order to make detection easier. [43]

The three most common algorithms to perform broad phase testing are *all-pair test* (exhaustive search) [43], *sweep and prune* (coordinate sorting), and *hierarchical hash tables* (multi-level grids). Exhaustive search is a naïve brute-force approach

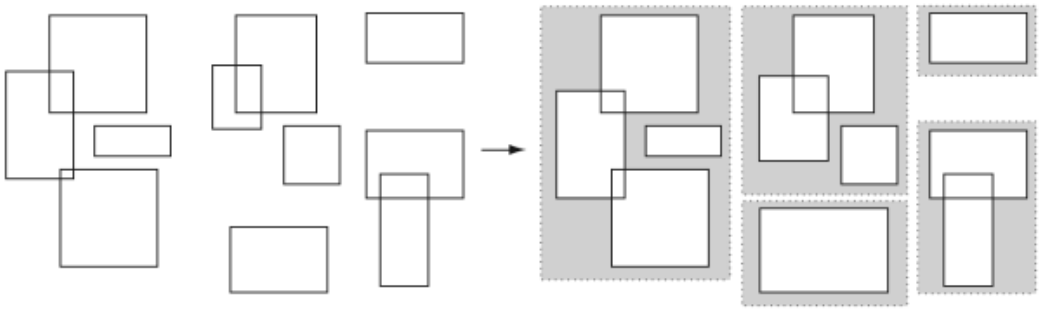

Figure 2: The broad phase identifies disjoint groups of possibly intersecting objects. (Image source [41])

which compares each object bounding volumes with others' bounding volumes. If volumes are intersecting, then the algorithm starts further investigation with narrow phase collision detection algorithm. Sweep and prune algorithm uses projection onto coordinate axes [44]. It projects every bounding volume's starting and ending points and checks if there is an intersection among all principal coordinate axes. Hierarchical hash tables is an another approach to speed-up computation [45]. It divides the entire scene into an equal size grid cells along all the principle axes. All grid cells containing multiple objects might potentially contain collision between those objects.

When the broad phase extracts all potentially colliding object pairs, narrow phase inspects further each of those pairs and determines if they really are colliding or about to collide. Narrow phase algorithms usually produce more detailed information, and are therefore much slower than broad phase algorithms. The gathered information can later be used to compute, for instance, time of impact, collision response and forces, and contact determination values. Some narrow phase algorithms return only a boolean value indicating if penetration is occurred [46]. They are sometimes called *interference detectors*. Narrow phase detectors that return the distance between disjoint objects are usually more useful, because the distance information can be used to more quickly compute the time of collision. In this work, the names of the narrow phase algorithms are mentioned and some of those are introduced briefly, but the more comprehensive survey of those methods can be found in [43].

Narrow phase algorithms can be divided into the four category: *feature-based*, *simplex-based*, *volume-based*, and *spatial data structures* [43]. Feature-based algorithms use the geometric primitives of the objects, such as spheres, capsules, or

even convex polyhedra [47]. Well-known examples of feature-based algorithms are *polygonal intersection* [48], *Lin-Canny* [47], *Voronoi-Clip* [49], and *SWIFT* [50]. Lin-Canny is the first feature-based algorithm mentioned in literature, and the last two algorithms mentioned are strongly based upon it. The *Gilbert-Johnson-Keerthi* (GJK) is a well-known simplex based algorithm [51]. It takes two sets of vertices as input and finds the Euclidean distance and closest points between the convex hulls. GJK is generalized to be applied to arbitrary convex point sets, not only to polyhedra. The third category is image-space based (ISB) algorithms. ISB techniques are computed by image-space occlusion queries which are effective to be executed on the graphics hardware (GPU). *Cinder* [52] and *CULLIDE* [53] are well-known examples of ISB algorithms. Volume-based algorithms are conceptually based on the same idea as the ISB techniques; however, they use different methods to compute layered depth images and distance fields. One example of such an algorithm is Gundelman [54], and in it all the objects are represented by a triangular mesh and the signed distance map.

2.3.3 Hierarchical volume bounding

In order to effectively check collision between complex objects, it is advisable to approximate objects with *bounding volumes* (BV) [55]. Often checking collision between BVs is enough to coarse determine possible collisions. Non-interference situations can be easily detected at the first levels in the *bounding volume hierarchy* (BVH), and more precise inspection is only necessary in the parts where collision may occur.

Bounding volume hierarchy is a tree-shaped structure, where the root (first level) is a coarse, one volume representation of the object [43]. A leaf, that is finest level of the tree, includes the object primitives such as lines, triangles, and tetrahedra, and between succeeding levels, there is a parent-child relationship, like in a tree topology.

The selection of bounding volume type depends on the usage. Various bounding volume types are presented in Figure 3. Often the type of objects is known beforehand, so the correct bounding volume is easy to select. If bounding volume must be selected without knowledge of the object size or shape, the more general shape is always better. However, the more general bounding volume is always more complex, and hence heavier computationally. The simplest solution is to use sphere or *axis-aligned bounding boxes* (AABB), due to the simplicity in checking two such volumes for intersection [57]. Another approach is to use *oriented bounding box* (OBB),

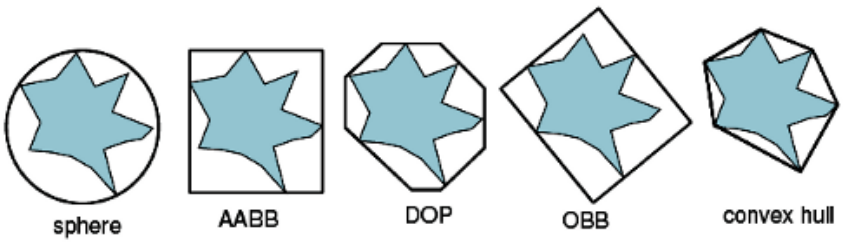

Figure 3: Different bounding volumes. To the left: smaller computational costs for overlap test are required. To the right: better approximation, but higher build and update costs. (Image source [56])

which surrounds an object with a hexahedron with rectangular facets, whose orientation is arbitrary with respect to coordinate exes. OBB can generally yield a better outer approximation of the object than AABB because its orientation can be chosen in order to make the volume as small as possible. Similar methods to OBB are *BoxTree* by Ballard [58] and Zachmann [59], and *OBBTree* by Gottschalk [60]. In Figure 3, there exists also k-DOP (where k=8) bounding volume, where DOP states for *discrete orientation polytopes*. An alternative name for DOP is the term *fixed-directions hull* (FDH), that is perhaps a slightly more precise[57]. *Convex hull* is an optimal solution to volume bounding problem and it provides the tightest possible convex bounding, but at the same time, it is the most expensive to compute [57].

2.4 Optical flow tracking

Efficient target tracking algorithms need some features to be extracted from the image. The most earliest techniques were based on edge extraction, with the most famous algorithms called *Sobel* and *Canny*. Corner points are another much used feature, and one popular algorithm using it is *FAST*. The most recent tracking algorithms are based on *optical flow*. It is the technique to get tracking algorithm focused only on areas where has been some motion between two successive image frames.

Optical flow is the representation of apparent velocities of movements of brightness patterns in an image [61]. Applying an optical flow algorithm on the image sequence brings out moving parts of the transient image. By comparing object velocities in 3-D world with optical flow in the image plane, one major problem arises; when an object is rotating, the object seems to be statical on image plane, and hence the optical flow all over the image is zero. In this application optical flow is used to

track the target in the image, so zero flow is not a problem. The technique behind optical flow is presented in detail in [61] and [62], but it is non-trivial and out of the scope of this work. In [63] different optical flow techniques are introduced and the performance is compared. Next, a couple of tracking techniques using optical flow are introduced.

2.4.1 KLT

KLT (*Kanade-Lucas-Tomasi*) is one of the first tracking algorithms based on an optical flow. The algorithm is based on optical flow method that was introduced by Lucas and Kanade in the early 1980's [64]. Although the algorithm has been improved in many ways since then [65, 66], it is still a basis for many later feature tracking algorithms.

2.4.2 TLD

TLD (*Tracking-Learning-Detection*) is a tracking method originally developed by Zdenek Kalal [67] during his PhD thesis. Kalal developed TLD using Matlab, but later, many implementations on other languages and frameworks have been done. The two most significant ones are probably the C++ version OpenTLD by Georg Nebehay [68] and Robot Operating System version of OpenTLD packed up by Ronan Chauvin [69].

The working principle of OpenTLD is presented in detail in the master's thesis by Nebehay [68]. In general, the main idea of TLD is that, in addition to tracking and detection, the algorithm contains a learning part that is capable of handling changes in pose and orientation as well as partial occlusions. The algorithm is autonomous, so that learning is done automatically after a model object is defined by an AABB for the algorithm. TLD is strongly based on the Lucas-Kanade method.

2.4.3 CMT

CMT (*Consensus-based Matching and Tracking of Keypoints*) is also award-winning object tracking algorithm by Georg Nebehay [70]. CMT is able to track a wide variety of object classes in a multitude of scenes without the need of adapting the algorithm to the scenario in any way. Where TLD uses template matching techniques to detect correct matches, CMT relies on keypoints extracted from the images. The paper itself does not take a stand on a method how the keypoints are extracted.

According [70], CMT outperforms TLD and other competitors when high accuracy is desired.

3 Collision detection system

As this thesis is a part of the larger project to extend a user's spatial awareness, some technical decisions have been made beforehand and are out of the scope of this part. One of those decisions is to use stereo visioning system to produce 3-D input data of the surroundings. Stereo vision is not in a major part in this thesis, and that section is discussed in details in [2]. Using stereo vision however causes some restrictions for the accuracy of input data, meaning that artifacts and other false objects might appear into input data. That kind of issues must be taken into account when new algorithms are implemented.

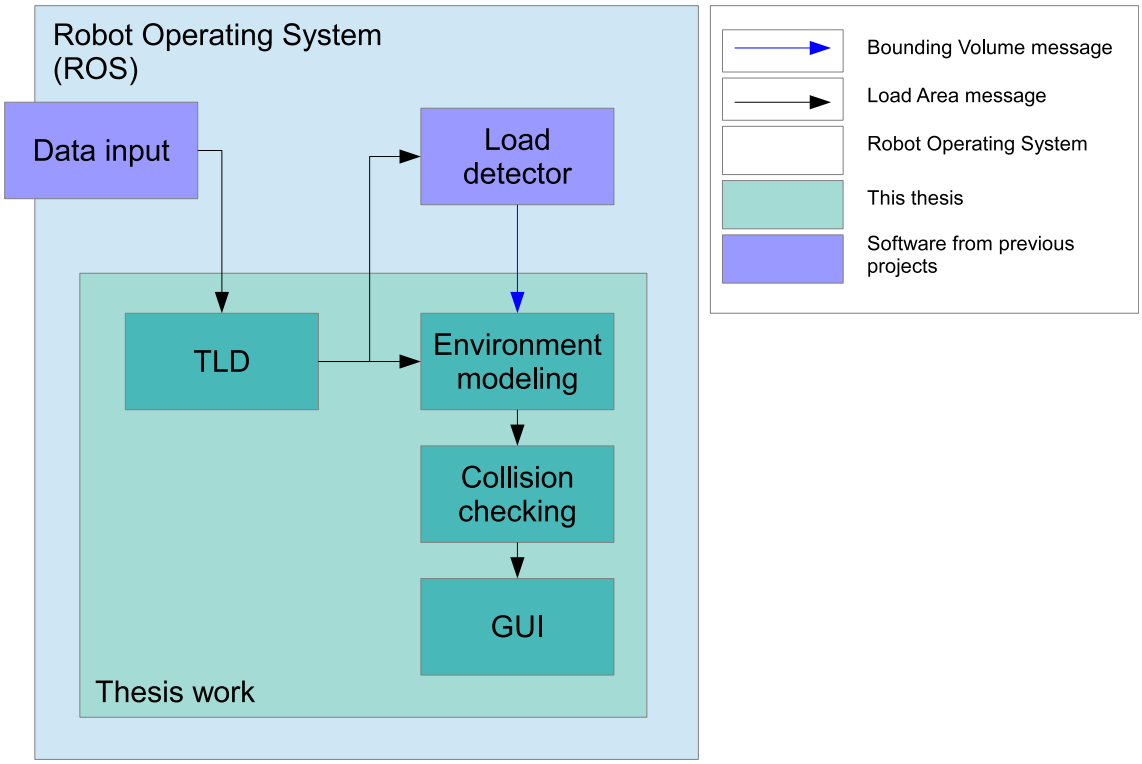

Figure 4: Software architecture of the collision detection system.

The architecture of the software developed during this thesis can be seen in Figure 4. The architecture was mostly decided beforehand and the selection criteria are also out of the scope of this thesis. The major choices included in this thesis are the environment modelling and collision detection algorithms, and the way how the results are presented for an operator.

ROS is used as a development platform and most of the software are executed on it. Application nodes with a blue color are developed in previous part of the whole project, and nodes with a green color are developed in this thesis. Arrows

express the data flow between each node. Data flow is based on *topics* and *messages* of the ROS network, in this case customized LoadArea messages are used. Blue arrow is expressing the Bounding Volume message containing the dimensions and 3-D coordinates of the detected load.

The TLD image target tracker is used to find the end-effector in the image fetch from the camera of the stereo imaging system. Having 2-D position of the end-effector, the actual 3-D position of the end-effector is calculated by combining the image coordinates and 3-D point cloud data fetch from the stereo vision system. Having the 3-D position of the tool, an actual collision checking is performed using a environment model built based on the point cloud data.

In this section, first, a short introduction to Software libraries and ROS as platform and TLD is given. Then the environment modeling algorithms is presented in details, and after that collision detection system using the presented model is introduced in details. In the final part the user interface for collision checking system is presented.

3.1 Software libraries

In this subsection, some ready-made software libraries for stereo vision and robotics system are listed. The presented libraries are selected mainly because they are the most commonly used and best supported libraries in their area.

3.1.1 OpenCV

OpenCV [73] is a well-known and widely used open source computer vision and machine learning library. It was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. It has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. They can be used to detect and recognize faces, identify objects, classify human actions in video, and much more. The library has C++, C, Python, Java, and MATLAB interfaces and it supports Windows, Linux, Android, and Mac OS.

In this work, OpenCV is used to process stereo image pairs, rectify them, and to calculate a range image based on the stereo image pair. Later, partly using OpenCV, also point cloud is generated from the depth images. As mentioned, OpenCV is part of the ROS framework, so it was a reasonable choice to image processing.

3.1.2 PCL

The *Point Cloud Library* (PCL) is a standalone, modern open source C++ library for 2-D and 3-D image and point cloud processing [74]. It manages 2-D or 3-D information about the environment as a set of the points. The point cloud is the easiest and one of the most efficient ways to store and handle 3-D information. The library provides a wide variety of algorithms to operate on pointcloud data. They can be divided into smaller sub-libraries that can be compiled separately. The list of the sub-libraries included in PCL is introduced in Table 1.

Table 1: The list of the libraries included in PCL.

Library	Description
filters	Filters like downsampling, outlier removal, indices extraction, projections
features	Many 3-D features such as surface normals and curvatures, spin images, integral images, RIFT, SIFT, etc.
io	I/O operations such as PCD (Point Cloud Data) file operations
segmentation	Cluster extraction, model fitting, polygonal prism extraction, etc.
surface	Surface reconstruction algorithms, meshing, novex hulls, etc.
registration	Pointcloud registeration methods like ICP, etc.
keypoints	Multiple keypoint extraction methods
range_image	Support for range images created from point cloud datasets
visualization	Visualization toolkit for several 2-D and 3-D data types

3.2 ROS Environment

The Robot Operating System (ROS) is a framework to work with robots [71]. Despite its name, ROS is a software package developed mainly for Linux, but also other operating systems have experimental versions. According to their web site [72], ROS contains »collections of tools, libraries and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms».

ROS contains a huge amount of features for different kind of task. The core components are ROS core, communication infrastructure and distributed parameter system. The communication infrastructure is responsible for transferring messages (ROS data packets) between different nodes (detached instances of ROS programs).

Different types of messages are used to transfer sensor data, images, pointclouds, control commands, and so on. Any node can easily subscribe a topic (socket interface that is identified by a string) in order to receive messages published by another node. *The Point Cloud Library* (PCL) and *Open Source Computer Vision Library* (OpenCV) are included in ROS libraries as well. These libraries are introduced in more detail later in this section.

In addition to the messaging framework and the large amount of libraries included in ROS, there exists also multiple diagnostics and monitoring tools. Just to mention some, *rviz* is a general purpose tool to visualize multiple three-dimensional message types and many sensor data types. By visualizing all the sensor data in the same application helps a user to quickly see what the robot sees. Another tool is *rqt* that visualizes the ROS network as a graph by drawing nodes, topics, and interconnections between them.

As the project has begun over 3 years ago, some libraries and softwares used are not up-to-date any more. One of those is ROS, for the old version *groovy* was chosen to be used. Newer version *hydro* contains many new and improved features, that are partly re-implemented during this project. Main difference between *hydro* and *groovy* are that *hydro* contains PCL 1.7 while in *groovy* the PLC version is 1.6.

Table 2: Terminology used in ROS.

Term	Description
Node	Distinct application or process
Topic	Communication interface between nodes
Message	Data packet transfered from topic to topic
Publish	Send data message from a node
Subscribe	Receive message to node

ROS infrastructure is not introduced here in details, but some terminology can be seen in Table 2. ROS contains several default message types for transferring, for example, integers, floats, images, markers, or point clouds. Messages contain always a header field that defines fields *frame_id* and *timestamp*. *Frame_id* specifies the point of reference for data contained in the message, more specifically, it is the name of the transformation between some fixed point in the environment and the data in the message. Messages are also always timestamped, so that it is possible to combine two different messages from different sensors based on a timestamp.

The nodes included in this project are using a custom message type called *LoadArea* that includes multiple message types, in this case point cloud, rectified image,

and the point of end-effector. The advantage of combining these data structures into a single message is that no time-synchronization between multiple topics is needed, but the disadvantage is that the same data is copied and transferred between all nodes and that increases the amount of data transferred inside the system. This might be a bottleneck if the computer used is not powerful enough.

3.3 Tracking the target

TLD (*Tracking-Learning-Detection*) algorithm was chosen to detect position of the end-effector, mostly because a ready-made ROS implementation for it was found. Also TLD includes a learning part that is important for the system as a pose of the object changes substantially. If the object was always with the same pose, it would be easy to detect its position with any method, but now the reliability is important as all the remaining parts of the system fully depends on the position of the end-effector.

As mentioned before, ROS version of the TLD was already implemented by Ronan Chauvin[69] so integration to system was quite straightforward. All that needed to be done was to modify input and output topics to LoadArea type. In addition the function to select 3-D point based on the tracked point on the image was implemented. This functionality was also quite simple, as LoadArea message contains the rectified image from the scene and point cloud generated from that (and the left-hand side image). The point cloud data is *ordered*, what means that data is in the array with equivalent dimensions compared to image used for tracking. When the tracked target was found, for instance, with the center point $p_c = (x_c, y_c)$ on the image, the corresponding 3-D point from the point cloud is then $P = (x, y, z)$. The point can be accessed directly using function $P = f(a, b)$, where $a = x_c$ and $b = y_c$. If the point at (a, b) has, for some reason, infinite distance (z component), the next one on the right is selected.

The TLD consists of two nodes where one implements the main tracking algorithm and the other is a graphical interface for initializing the tracker. The tracker can be initialized also with a configuration file and a made-up model. The model contains all data needed to track a target, so it is possible to generate the model beforehand and run the algorithm autonomously without any user interaction.

3.4 Environment modelling

In the begin of the project, there was a need for a 3-D model of the dynamic environment. Constructing the model would be simple since the cameras were stationary and only the environment was changing. No simultaneous localization and mapping is needed. In a previous thesis in this project[1], the collision detection libraries for point cloud and octree were implemented. Because the environment is partly dynamic, and some occlusion exists, octree with a knowledge of free, unobserved, and occluded cells was selected. However, during this project it turned out that constructing the 3-D octree was far too expensive operation for the computer hardware and libraries present. This is discussed in more detail in the next sections, but now, the much cheaper modeling algorithm is introduced.

The algorithm implemented is based on a two-dimensional array (later referred as *model*). Using only two dimensions (like a depth image) does not lose any important data, but reduces the amount of data significantly. The reduction is based on a probabilistic nature of the algorithm. In order to initialize the model, the resolution of the grid must be determined beforehand, because the size of the grid is static and computed based on the given bounds of the environment. In the grid, each cell has two floating-point values, a distance and a probability. The distance value is updated always when new data is available, and updating the probability depends on whether a cell is occupied or not. The incoming 3-D data must be processed in advance, before it can be updated into model. The processing steps include data downsampling, bounding into the limits, and reprojection on the 2-D plane of the model. In the next subsections the steps to create and update the environment model are presented.

3.4.1 Data filtering

Depending on the input data, pre-filtering is often necessary. First the given input data must be bounded into some predefined limits. Especially if the z-value (distance) of a data point runs over a given limit, it is recommend to extract it from the dataset in order to accelerate processing. If the input dataset is enormous, it can be down sampled to given a resolution. Using that resolution, the dimensions of the model is computed so that no extra space is required and also data will not vanish. If dataset is very noisy, it can be low-pass filtered, although low-pass filters provided by PCL 1.6 are currently quite computationally expensive. Later, segmentation of the input dataset will be performed, which also filters small noisy areas out of the

dataset, hence low-pass filtering is not necessary.

As mentioned before, parts of input data can be cut out based on some rules. In this project, all dynamic objects will be removed from the input data. They are a boom, an end-effector, and a possible load. As can be seen in Figure 4, environment modeling node has all the information needed to extract those parts from the dataset. The *TLD* node provides the 3-D point of the end-effector and *Load detector* provides a pose and dimensions of the load. Using these positions, axis-aligned bounding box (AABB) is fitted to include those dynamic objects.

3.4.2 Segmentation

In order to include the boom and end-effector successfully into AABB of dynamic objects, the point cloud must be segmented. The segmentation extracts points nearby into Euclidean clusters using *Euclidean Cluster Extraction* algorithm from PCL. The algorithm is simple and it does not take account any other measures than distance to the points nearby. PCL 1.7 provides also other segmentation methods, but they are not implemented in the version 1.6.

The Center points of the each segmentation clusters are computed and nearest cluster from end-effector point (from TLD) is selected. AABB of that cluster is then included into AABB of the dynamic objects. Then all dynamic objects are extracted from the input data, so that the remaining pointcloud contains only static objects.

3.4.3 Environment Model

The model used here can be seen as an *extended probability grid* or *extended depth map*. It differs from occupancy grid, because occupancy grid only keeps track of the probability that cell is occupied, but in this model also z-value is present. More generally, normal occupancy map is for two-dimensional mapping, but this model allows cheap 3-D mapping for a static viewer case.

The representation format of this probability grid is a fixed size two-dimensional array $A(i, j)$, where each cell is a pair $a_{i,j} = (P_{i,j}, d_{i,j})$. The values stored are the probability $P_{i,j}$ that the *ray* is occupied and the distance $d_{i,j}$ to the object.

The resolution value for down sampling the input data is preconfigured and it is also used to determine dimensions of a probability grid.

The size vector of the array containing also a maximum depth value is computed

using

$$\mathbf{S} = \begin{bmatrix} \lfloor \frac{w}{R} \rfloor \\ \lfloor \frac{h}{R} \rfloor \\ 255 \end{bmatrix} \tag{6}$$

where w is width, h is height of the limited environment and R is a given resolution. Note that fractions are rounded down towards the nearest integer. Third value of the size vector is the maximum value of the scaled depth, as the algorithm is using 8-bit unsigned integers to represent depth. As an example, if the environment is limited inside a box with a dimensions $20 \times 20 \times 10$ meters and resolution is selected to be 0.02 (meters per cell), the dimensions of the array grid would be

$$\mathbf{S} = \begin{bmatrix} \lfloor \frac{20}{0.02} \rfloor \\ \lfloor \frac{20}{0.02} \rfloor \\ 255 \end{bmatrix} = \begin{bmatrix} 1000 \\ 1000 \\ 255 \end{bmatrix}. \tag{7}$$

Using 8-bit integers, the depth resolution for 10 meters will be $\frac{10}{255} \approx 4$ cm, what is enough for the most cases.

Updating the probability P of the cell $a = (i, j)$ in probability map is done with a following formula

$$P(z_{k+1,a}) = \begin{cases} P(z_{k,a}) \cdot P_{post} + P_{prior} & \text{if } a \text{ is occupied} \\ P(z_{k,a}) \cdot P_{post} & \text{if } a \text{ is free} \end{cases} \tag{8}$$

where a is the cell in the model, z_k is current model, and z_{k+1} is the updated model. Probabilities are chosen to be constant values $P_{prior} = 0.2$ and $P_{post} = 0.7$. The probability values were chosen after testing the implemented algorithm and by noticing that results are quite stable if they are somewhere near the given values. Based on the updated probabilities, new distance values can be merged into model using following formula

$$d(z_{k+1,a}) = \begin{cases} \hat{d}(a) & \text{if } \hat{d}(i) > 0 \text{ and } P(z_{k+1,a}) > P_{limit} \\ d(z_{k,a}) & \text{if } \hat{d}(i) = 0 \text{ and } P(z_{k+1,a}) > P_{limit} \\ 0 & \text{else} \end{cases} \tag{9}$$

where $d(a)$ is the distance value at the cell a in the model, and $\hat{d}(a)$ is a measured distance value so that $(0 \leq d(a) \leq 255)$. P_{limit} is reconfigurable constant to determine threshold value for filtering out *improbable* areas from the model.

Model memory requirements can be roughly approximated using array dimensions and stored variables. Memory requirement for one cell is a 8-bit integer and a 32-bit float, in total 40 bits. Using 1000×1000 grid, the total memory requirement of 40 Mbit $\approx$ 5 MB can be obtained.

This model is suitable especially for cases where the environment is partly moving, for instance, trees are swaying or an extra object comes suddenly onto scene, but where the viewer is static during the inspection. The model is not able to differentiate two successive located objects, like an object behind a wall, but the same problem occurs for statically located sensor such as laser scanner. In this project the occlusion is not in a significant part.

3.4.4 Reprojection

In order to present data in a two-dimensional array, reprojection from 3-D space to 2-D plane is needed. Reprojection is done by moving the origin of the coordinate axes from the center of the camera point to upper left corner of the bounded environment (see Fig. 5). After that, each dimension is scaled using predefinied resolution value R. Affine transformation matrix T is constructed from the scaling M and translation T_v parts using the formula

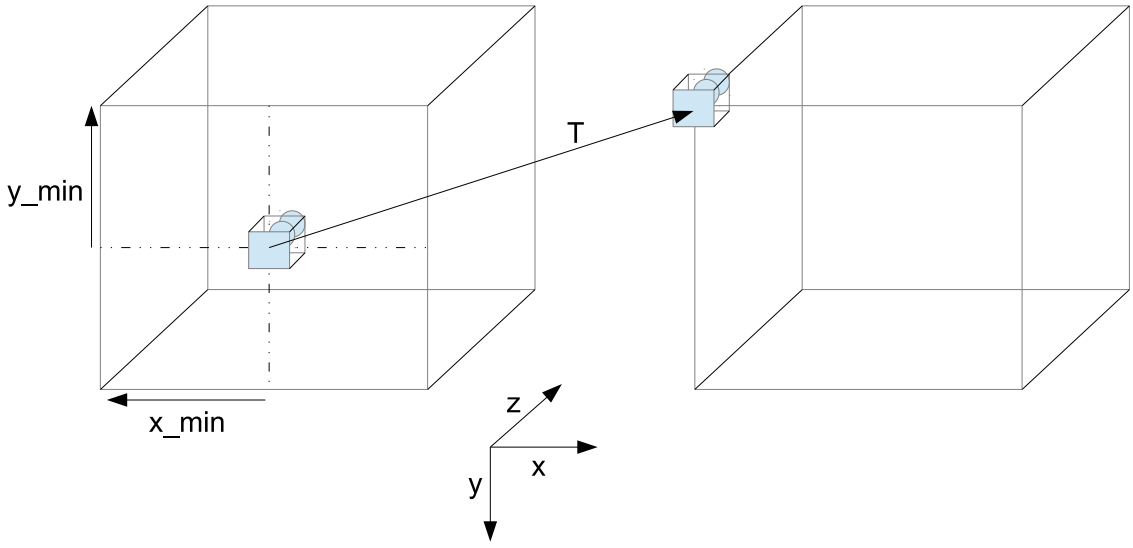

Figure 5: Coordinate transformation T from 3-D to image plane.

$$T = MT_v = \begin{bmatrix} \frac{1}{R} & 0 & 0 & 0 \\ 0 & \frac{1}{R} & 0 & 0 \\ 0 & 0 & \frac{255}{z_{max}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & x_{min} \\ 0 & 1 & 0 & y_{min} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{10}$$

where R is constant resolution and x_{min}, y_{min}, and z_{max} are minimum and maximum values of bounded environment, respectively.

Having T, transformation from a homogeneous 3-D point $P(X, Y, Z)$ onto a image plane $p(x, y, z)$, where z is a scaled depth value of the cell $a(x, y)$ of the array A, can be obtained using a formula

$$p = TP = \begin{bmatrix} \frac{1}{R} & 0 & 0 & \frac{x_{min}}{R} \\ 0 & \frac{1}{R} & 0 & \frac{y_{min}}{R} \\ 0 & 0 & \frac{255}{z_{max}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}. \tag{11}$$

Now all that has to be done is to check that values (x, y) of the p are in the bounds of model dimensions, because values were rounded down to nearest integer (see the Eq. (6)) when the model dimensions were determined. When updating the model, a cell $a(i, j)$ values i and j are the same as the p's values x and y and distance $\hat{d}(a) = z$. So in the cell a where $a(i, j) = a(x, y)$, the distance $\hat{d}(a) = z$.

3.5 Collision checking

Collision checking could have been done using ready-made libraries such as Open Dynamics Engine (ODE), but they are using 3-D environment representation formats like point clouds or octrees to perform collision detection and now created model is just a two-and-a-half dimensional. Also, in this project, there is only one object to check collision with, so implementing a new simpler library to perform collision checking was reasonable.

Collision checking is done between the environment model generated using the methods above, and the object enclosed by OBB (oriented bounding box). Using OBB allows much more accurate fitting than using AABB (see Fig. 6), and because the object in this case is a cylinder shaped load, the fitting will be exact. First of all, the 3-D points of the OBB of the load must be transformed into model coordinates. This is done using the same affine transformation (see Eq. (11)) as

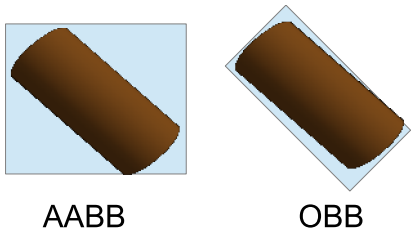

Figure 6: Axis-aligned and oriented bounding boxes encapsulating a load.

when transforming a model. Bounding volume can be represented using just four corner points.

3.5.1 Two-phase checking

Instead of a broad and a narrow phase mentioned in Background section, this algorithm requires just a second phase for selecting a possible colliding object, because there is just one object to collide. However, the developed algorithm is divided into two phases: first phase is for checking collisions behind the object, and second phase is for checking collisions around the object. The first phase is minor part of collision checking and might be omitted especially with small or narrow objects, because it is very unlikely that there is a small piece of environment *only* behind the object. The second phase is the major part of this algorithm. It checks possible oncoming collisions in all directions around the object. Collision is checked along the rays drawn from the object (see Fig. 7), and nearest point between the environment and the object is calculated.

3.5.2 Functionality

The functionality of the first phase is simple. All cells inside the bounding volume are gone through and if any cell contains a piece of environment, the nearest point from the bounding volume is sought. The method to calculate the euclidean distance between two cells is presented in a next section. At the first phase, the distance is simply a scaled difference between bounding volume and the environment along the z-axis.

The second phase is a bit more complex as the Euclidean distance must be calculated over all three axes (see Fig. 8). First, the search rays are computed and and each cell along those rays are inspected. Rays are *normals* of the bounding

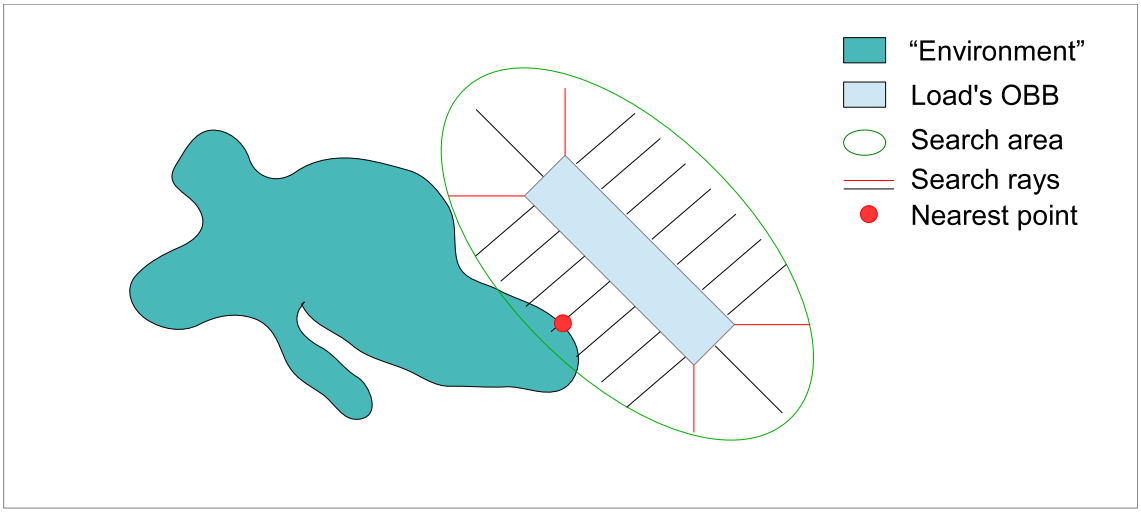

Figure 7: Figure shows the concept of collision checking on a model plane. In Figure, the environment is demonstrated as a hand drawn area and the object's bounding volume is shown as it could be at a simple situation. Search rays from the corners are visualized with a red color, and from the sides of the OBB with a black color. Rough search area is visualized with a green ellipse and a nearest point on the "environment" measured from the load is shown as red dot. Although the collision checking is performed on the plane, the depth values of each cell are taken into account, so the checking is actually performed on the two-and-a-half-dimensional space.

volume, so if the bounding volume is rotated around the x-axis so that the end of the bounding volume is *further* away than the other, rays are also at a different *distance* or they can even be rotated around all the axes. The interval of the search rays can be configured according to a size of the particles in the environment. If the single particles are small, then the interval of the search rays should be small enough. Also the *search radius* (radius of the search area) is configurable and can be adjusted, for instance, according to the speed of the observed object.

3.5.3 Vector arithmetics

The bounding box, corners, sides, search rays and distance calculations are handled with simple vector arithmetics. The corners of the bounding box are named as m_i where $i \in \{1, 2, 3, 4\}$. Each side of the bounding box is then $s_i = m_j - m_i$, where $j = i + 1$ and if $j > 4$, $j = 1$. Next, select each cell (point) inside the bounding box by selecting two sides. The first side s_1 is a baseline for an examination. Each point

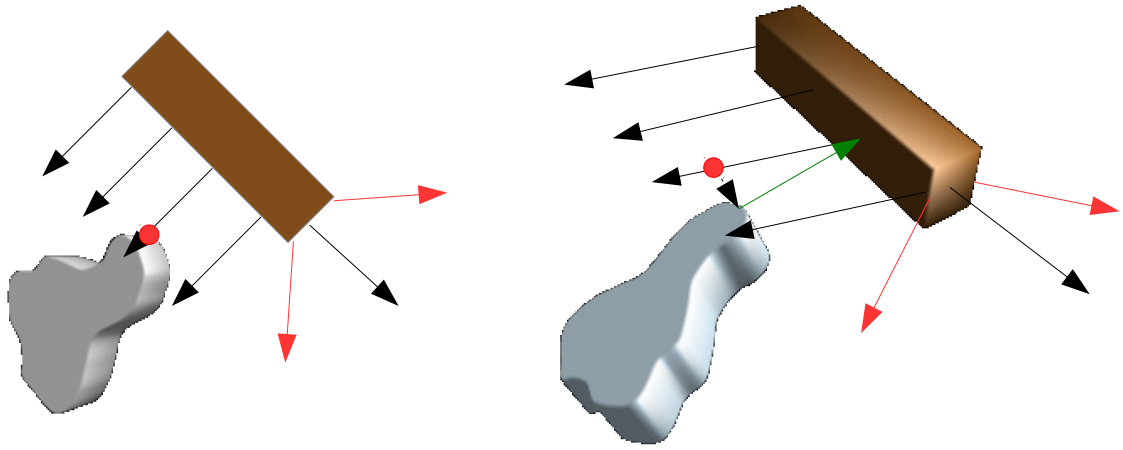

Figure 8: Figure shows from two different points of view, how the distance between the environment and a load is calculated. The green line is the correct Euclidean distance between the environment and a load, and a red circle is the nearest point on the search ray. Only part of the search rays are visible.

$\dot{p}_r$ on that side can be reached using a formula

$$\dot{p}_r = m_1 + r\frac{s_1}{||s_1||} \tag{12}$$

where the last term is a normalized unit vector of s_1 multiplied by an index of the point r, where $r < ||s_1||$. Now, using the vector of the other side, each point inside the bounding box can be obtained by using a formula

$$p_{r,s} = \dot{p}_r + s\frac{s_2}{||s_2||} = m_1 + r\frac{s_1}{||s_1||} + s\frac{s_2}{||s_2||} \tag{13}$$

where $r < ||s_1||$ and $s < ||s_2||$. Vector arithmetics for accessing each point inside the bounding box is demonstrated in Figure 9. The nearest point inside the bounding box is the point where $\min ||p - p_{r,s,Z}||$ is true, and p is an environment point at the same location as the bounding box point $p_{r,s}$. The distance between those points is a distance along the z-axis.

Accessing points around the bounding box is done similarly. Instead of using s_2 to move *inside* the area, the normal vector of each side is used to move along a search ray. For the side s_i, the normal vector is

$$n_i = -\frac{s_j}{||s_j||} \tag{14}$$

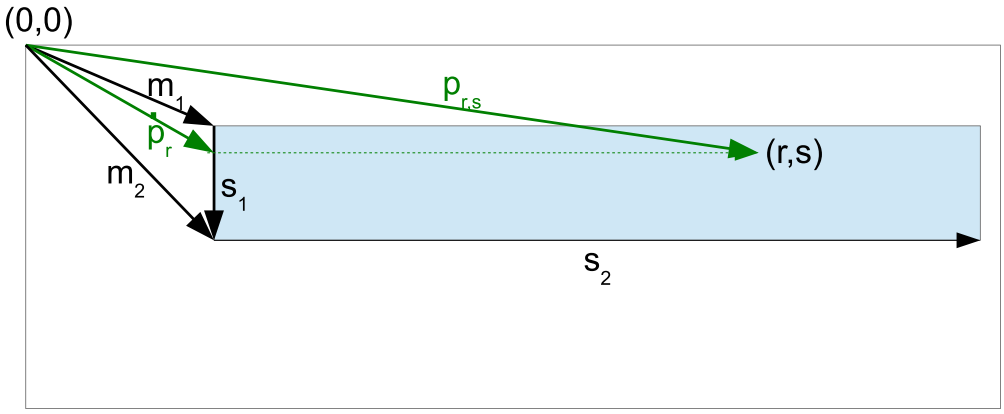

Figure 9: Vector arithmetics of the bounding box area.

where $j = i + 1$ and if $j > 4$, $j = 1$. Note that the normal vector is an negated unit vector of the *next* side of the bounding box. As every point around the bounding volume is not intended to be accessed, an interval value I is used to skip a part of the points on side lines. Every I'th point $p_{i,k}$ on the sideline i can be computed using a formula

$$p_{i,k} = m_i + kI\frac{s_i}{||s_i||} \tag{15}$$

where $kI < ||s_i||$ and $k \in \{0, 1, 2, 3...\}$. Now, using $p_{i,k}$, L points along the normal vector are selected using a formula

$$p_{i,k,l} = p_{i,k} + l \cdot n_i = m_i + kI\frac{s_i}{||s_i||} - l \cdot \frac{s_j}{||s_j||} \tag{16}$$

where $l \in \{0, 1, 2, ..., L\}$. i is an index of the bounding box side, k is an index of a ray on that side, and l is an index of the point on that ray. Computing the search rays is visualized in Figure 10.

Corner rays are computed similarly. The direction of the corner ray is computed using both side vectors connected to the corner. For the corner point m_i, the direction vector u_i can be written as follows

$$u_i = \frac{s_j}{||s_j||} - \frac{s_i}{||s_i||} \tag{17}$$

where $j = i - 1$, but if $j < 1$, $j = 4$. The unit direction vector $\hat{u}_i$ is

$$\hat{u}_i = \frac{u_i}{||u_i||} \tag{18}$$

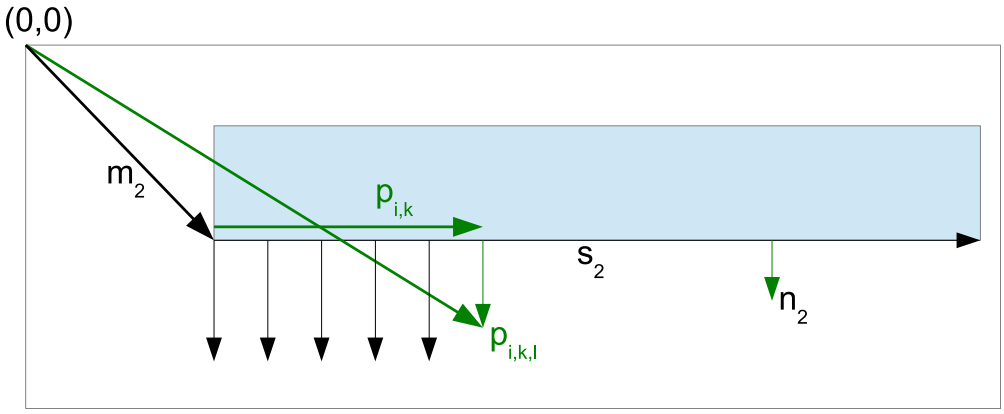

Figure 10: The figure visualizes how the outer search rays are computed using simple vector arithmetics. The point $p_{i,k,l}$ in this case is a point where $i = 2$, $k = 5$, and $l = 20$, if a search interval I is defined to be 10 and a search area L is defined to be 30. n_2 is the normal vector of the side s_2.

Using Equations (17) and (18), the length L corner search vector can be computed. Let $p_{i,l}$ be a point on the corner ray at the corner i. The point can be written as

$$p_{i,l} = m_i + l\hat{u}_i \tag{19}$$

where $l < L$. The nearest point on the corner ray can be found similar way as the other search rays. In order to calculate a scaled distance correctly, the resolution of the model must be known.

Suppose that p_e is a point on the environment and p_r is the point on the ray. The distance d between two points is defined to be $d = ||p_r - p_e||$. However, when the model was constructed, the points were scaled with different constants along the different axes, to be precise, a ratio along the x and y-axis is the same R (resolution), but along the z-axis it differs ($z_{max}/255$). In order to calculate the correct distance between two points, each component of the point must be scaled separately. The scaling matrix S_v is defined as follows

$$S_v = \begin{bmatrix} R & 0 & 0 \\ 0 & R & 0 \\ 0 & 0 & \frac{z_{max}}{255} \end{bmatrix} \tag{20}$$

where R is the configured resolution of the environment model and z_{max} is the maximum depth of z-axis in meters. The distance between two vectors is then

$d = ||S_v(p_r - p_e)||$, and it can be expanded as

$$d = \left|\left| \begin{bmatrix} R & 0 & 0 \\ 0 & R & 0 \\ 0 & 0 & \frac{z_{max}}{255} \end{bmatrix} \begin{bmatrix} p_{r,x} - p_{e,x} \\ p_{r,y} - p_{e,y} \\ p_{r,z} - p_{e,z} \end{bmatrix} \right|\right| = \left|\left| \begin{bmatrix} R(p_{r,x} - p_{e,x}) \\ R(p_{r,y} - p_{e,y}) \\ \frac{z_{max}}{255}(p_{r,z} - p_{e,z}) \end{bmatrix} \right|\right|. \quad (21)$$

The problem of finding the nearest point can then be condensed into a clause: find $\min d$.

3.6 The user interface

This section presents the software and user interfaces implemented for the given algorithms from previous sections. The windows of the software are created using PCL's and OpenCV's visualization functions. The 3-D view of the point cloud (see Fig. 12) is created using PCL visualization plug-in and the other two (see Figs. 13 and 14) using the OpenCV.

The demo program for testing the implemented collision detection algorithm using a manually generated environment model is visualized in Figure 11. In the window the gray floor is generated manually so that the *floor* is a bit forward-slanting. The brighter the floor (environment) is the nearer it is. In the center of window is the bounding volume of the object. The bounding volume is bordered with a yellow and blue colors and all green points are the points where the search algorithm is performed. In this demo, the search area is defined to be 80 pixels and the search interval is 5. Two nearest points are found, one inside the bounding volume (which is actually pointless) and one right below the bounding volume. The distance from the bounding volume to the red dot is visualized in upper left corner of the window. Red bar shows distances nearer than two meters and when the bar is full, the collision is occurred.

The environment modeling software receives input data as a point cloud. In the software, the data is limited inside configured bounds and the cloud is segmented. Small and noisy areas are filtered out and rest of the data is updated into environment model. Before that, bounding volume and dimensions of the target object are received, and segmented clusters inside and near the bounding volume are cropped off (in Figure 12, the area with red frames). Next, the point cloud is transformed onto a 2D plane of the environment model and probability map of the model (see Fig. 13) is updated. Note that in probability map, the machine boom and the load are not present.

When the probability map is updated, the actual model will be updated. The

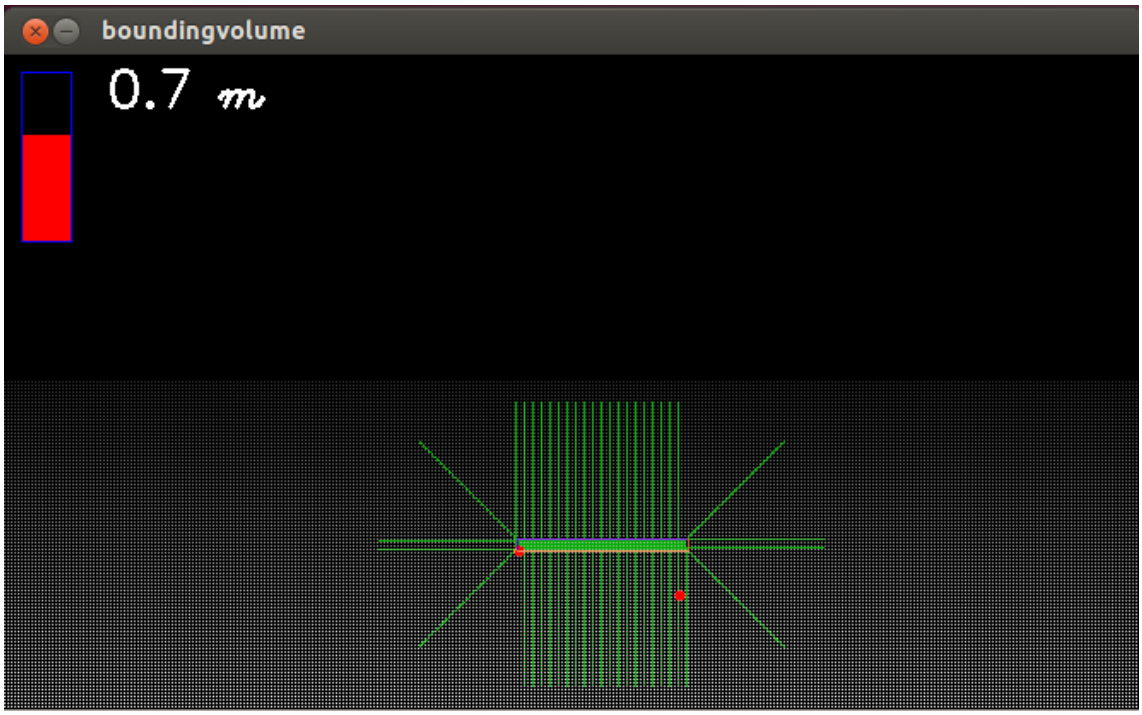

Figure 11: Collision detection demo. The floor (environment) is generated manually, and a bounding volume is moving along all three axes during the demonstration. Distance from the bounding volume to nearest point (marked with a red dot) is visualized in upper left corner using a bar and decimal number.

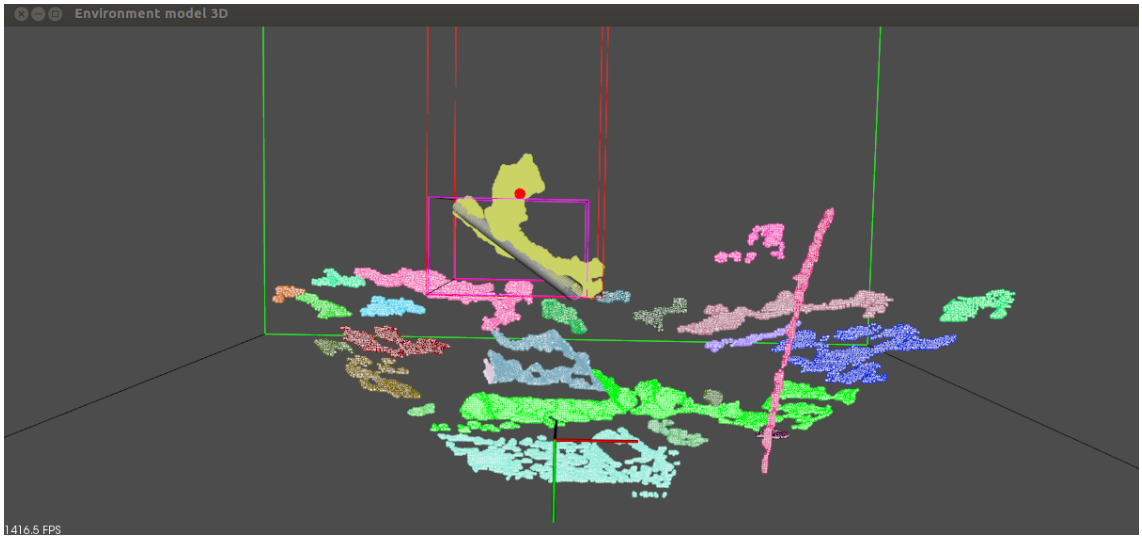

Figure 12: Input point cloud data, which is filtered, bounded and segmented before visualization. Target object and its violet bounding box is on the center of window. Red sphere encompasses all the dynamic objects on the scene. Green sphere visualizes the limits of the environment.

updating is based on a reconfigurable tresholding value, so that the most improbable areas are excluded. The environment model corresponding the point cloud seen in Figure 12 can be seen in Figure 14. After the updating process, the collision checking can be performed. The colliding object is first projected on the same 2D plane as the model and the nearest point is computed using the algorithm presented in a previous subsection. Collision checking is performed along the green rays visualized in Figure 14 and the red dot shows the nearest point found.

Parameter configuration is done using a graphical *rqt_ reconfigure* tool provided by ROS. It provides a way to view and edit parameters that are accessible via *dynamic_ reconfigure* plug-in from the ROS nodes. Reconfigurable parameters in this case are listed in Figure 15. First six parameters are the bounds of the environment. Note that the PCL uses right-hand coordinates, so positive y goes downward, and the *camera* is on the origin looking forward along the positive z-axis. *Z-reprojection* is not used in this case, but it would allow to reproject all the filtered points to a distance of that value. Low-pass filter is an optional feature, and next five parameters are connected to it. *Resolution* is one of the most important values this configu-

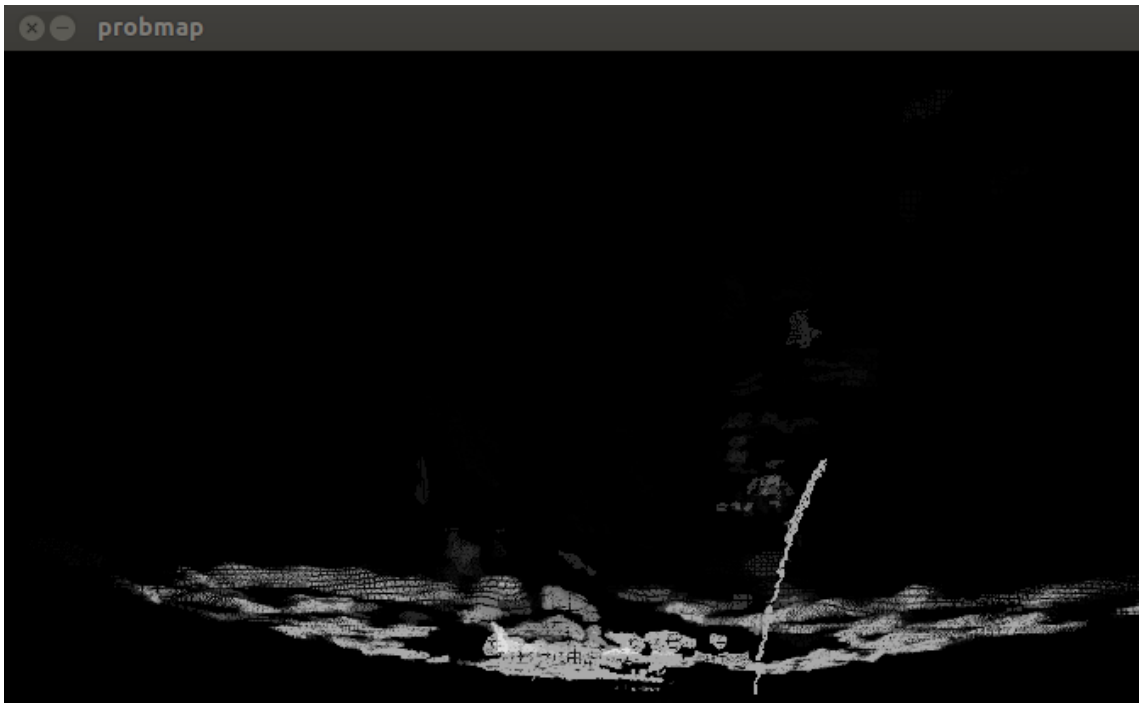

Figure 13: Window shows probability map of the scene. The model is constructed from the points where the probability is large enough. Dark areas are tresholded out from the model. The probability map is constructed from the scene seen in Figure 12.

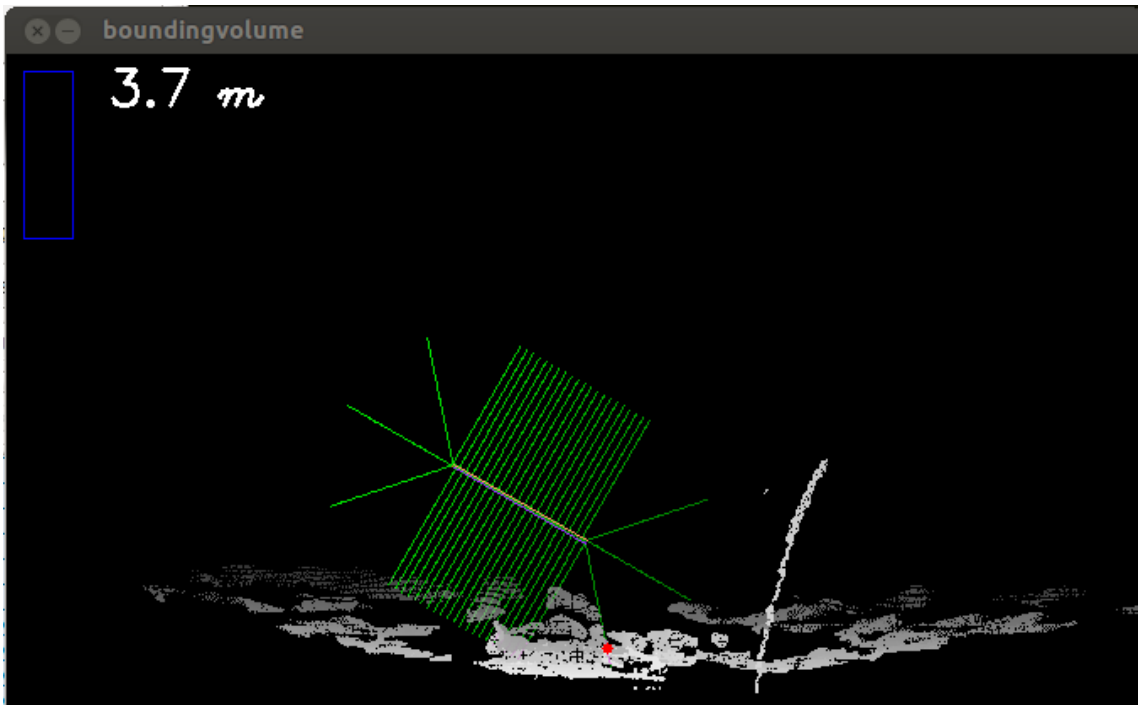

Figure 14: Environment model and collision checking using the model generated based on the probability map in Figure 13. Bounding volume here is the same as in a Figure 12. Nearest distance from object to ground is 3,7 meters.

ration tool provides. It determines the size of the environment model as explained in previous section. Another important parameter is *probtresh*, which is tresholding value to exclude improbable areas from the environment model. These two values (and first 6) must be adjusted according to the input data and the use case.

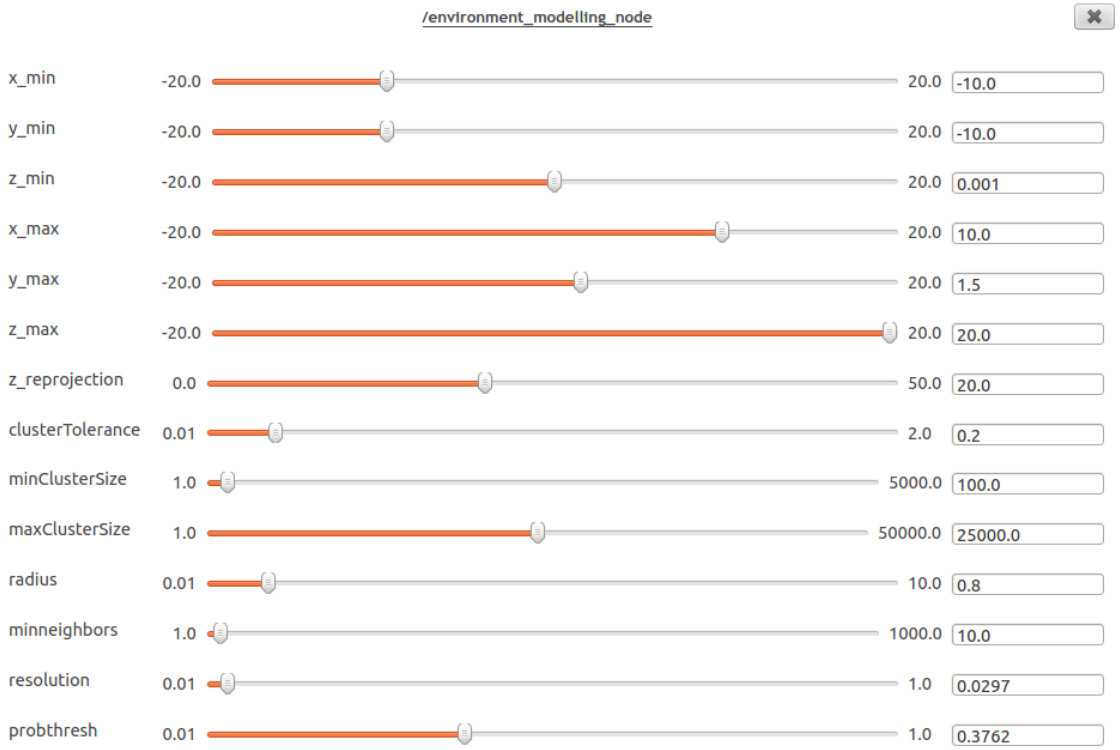

Figure 15: Dynamic reconfigure view (rqt_reconfigure) of configurable parameters.

4 Experiments and results

In this section the developed collision detection system is studied. The goal of the experiments was to show that the developed collision detection system is able to perform on the real environment and it is accurate enough to be used to extend operator's spatial awareness. The goal was also to show that implemented algorithm is easy to reconfigure and to include into the existing system.

The results are divided into three parts. In the first part the accuracy and the execution times of the tracking algorithm TLD are presented. Next, the results of an environment model generation and the update process are discussed. The results contain datasets recorded using Microsoft Kinect, which produced the most accurate 3-D data available. In the final part the results of the implemented collision detection algorithm are presented. The experiments are performed using the developed simulator, since the real environment tests were unavailable.

The hardware used for testing consists normal PC parts listed in Table 3. All the applications required for testing are single threaded and executed on the same computer at the same time. The computer is not connected to stereo imaging system, but incoming data is read from image files on a hard disk and a 3-D point cloud is computed locally.

Table 3: The hardware of the test computer used.

Dataset	Description
Model	Fujitsu ESPRIMO E910 E90+
CPU	i5-3470 CPU @ 3.20GHz
Memory	8 GB DDR3 1600 MHz
HDD	SATA III, 7.200 rpm, 500 GB
OS	Ubuntu 12.10

4.1 Target tracking

TLD (Tracking-Learning-Detection) algorithm was selected mostly due its ability to learn new poses of the tracked object. As mentioned before, TLD is used to track the position of the manipulator tool in the 2D image, and because the tool

can rotate along its axis, the silhouette of the tool changes. In order to track the tool successfully during the rotation, the new models must be learned. The most important reasons to select TLD, was that during the development of this system, it was one of the leading and the most successful methods to track objects. Also *Robot Operating System* version was available for it, so integrating it to the rest of the system was quite easy.

Four datasets were used to study the performance of TLD algorithm. In Table 4 the description of the datasets can be seen. The end-effector is at least partly visible in every frame, but a weight is loaded only in the datasets 1, 3, and 4. Because the actual screen captures and the images describing the experiment area were forbidden to be published by the owner of the area, a drawing from test scene is present instead in Figure 16. The drawing is 2-D illustration similar to the scene used while recording the datasets.

Table 4: Description of the datasets.

Dataset	Description
1	Simply move tool over the target, grab and lift. Target is visible during the period. The model of the target is not trained for this specific case, so accuracy is quite low.
2	Swinging, moving and rotating tool all over the scene. Sometimes tool swings so much that it is partly occluded by itself and tracker loses it for a few frames.
3	Simply rotate tool while holding a load. The tool is occluded by the load in one frame.
4	The most complex case. The load is moved all over the scene and tool disappear from the scene in 46 frames.

All the datasets are from one test scene recorded on the same day. The tracker is trained using a single model of the tool and it has automatically learned different tool poses from couple of datasets. When obtaining the results, the learning mode is turned off and tracker uses only knowledge of previously recorded poses of the target. By turning off the learning mode, the tracker will not learn any right or wrong poses and by that the TLD will not "explode", meaning that it will not start to learn wrong targets and little by little replace all the correct models by false positive ones.

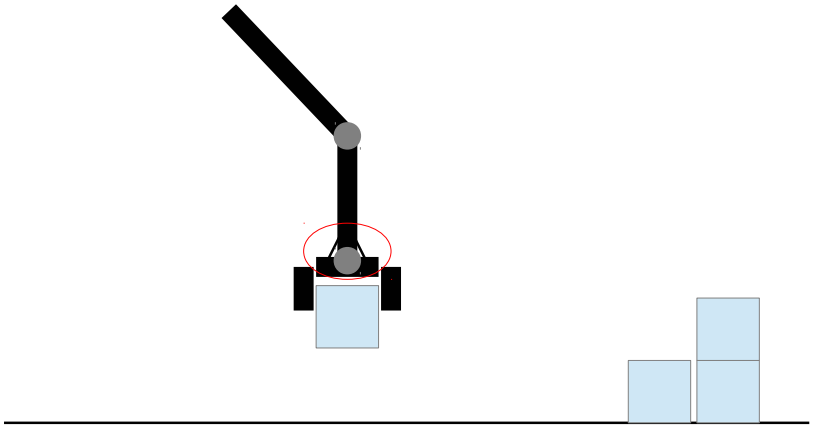

Figure 16: 2-D illustration of the test scene similar to datasets used to compare TLD results. The red circle visualizes the boom tip, the TLD's model target used for tracking.

4.1.1 Results

In Table 5 the individual results of four datasets are listed and the means are computed. The columns *Frames*, *Lost*, and *Occluded* present the number of video frames in the datasets, the number of lost tracking frames, and the number of occluded target frames respectively. The columns *Lost (%)* and *Accuracy (%)* are the percentage ratio of the lost frames compared to total frames, and the accuracy of the tracker when occluded frames are removed. FPS is the average number of a frames computed per second ratio for tracking the target. It is computed by calculating the average of the FPS $1/t$ of each frame. t is the time for tracking a target in a single image.

Table 5: Results of testing TLD for multiple datasets.

Dataset	Frames	Lost	Lost (%)	Occl.	Acc, (%)	Mean	Std	FPS
1	46	0	0	0	100.0	0.6464	0.0581	18.33
2	334	38	8.0	0	92.0	0.5570	0.2102	17.92
3	475	1	0.2	1	100.0	0.7068	0.0738	18.14
4	531	55	10.4	46	98.3	0.6438	0.1155	19.34
	1386	94	6.8	47	96.6	0.6385	0.1144	18.43

The results show that the algorithm is fast, being able to process about 18 frames per second, which will be enough for the most real-time cases. The accuracy of the TLD in this application is high, almost 100 percent if the dataset 2 is excluded. In the dataset 2 the tool is sometimes occluded by the boom and it is not totally

comparable with the other datasets. Even including the dataset 2 into results, the average accuracy rises up to 97 percent.

4.1.2 Discussion

During the tests, TLD was found to work very well, and there was no major reason to improve it. While finishing the writing this thesis, a new tracking algorithm CMT (*Consensus-based Matching and Tracking of Keypoints*) was published [70]. According the paper, it should be improved from TLD, and as future work, it would be interesting to check if CMT outperforms the TLD in this application.

4.2 Environment model

Multiple environment modeling and representation methods were presented in the Background section of this thesis. The basic methods are well studied and they have been used for years in many projects. The demands for the environment representation algorithm during this project were ability to store 3-D information of the environment, fast updatability, and knowledge of history to prevent dynamic objects to be updated into model. *Octree* had probabilistic nature, and it was able to store 3-D data, but during the initial tests, it was far too slow to be updated in real time. Update times exceeded several, even dozens of seconds. Because in this project the stereo cameras are statically mounted, and objects in the scene can be seen only from one side, the model can be also simplified to store "$2\frac{1}{2}$-D" information of the environment. In order to improve memory usage, a new modeling method was developed.

Environment modeling relies significantly on the accuracy of the incoming point cloud data. To test only modeling part of the whole project, some reliable input data is needed. Because no accurate 3-D laser range finders were present, Microsoft Kinect was chosen to be used. Kinect can produce accurate and dense point cloud data, that is easy to use by a ROS application. All the screenshots of an environment model and a probability map presented in this section are captured from the real application, but the colors have been inverted in order to clarify those on a printed media.

The environment modeling algorithm presented in this thesis is able to store simplified 3-D information similar to method known as a depth image. Only one side of the scene is recorded and stored into a model. Having stationary sensors, dynamic contents can be filtered out using knowledge of the previous frames in the model. In

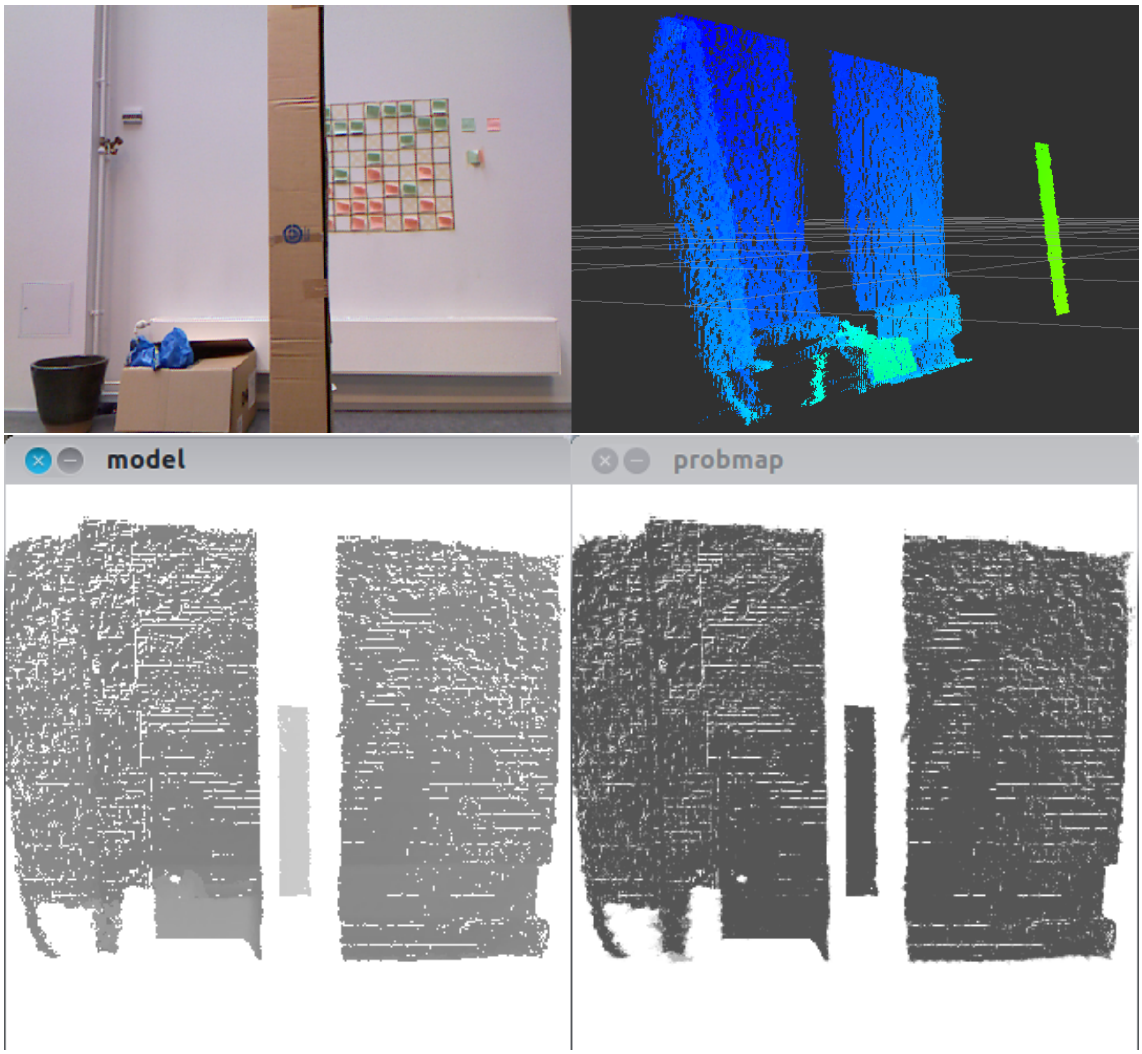

Figure 17: Simple scene showing the functionality of the environment modeling algorithm in practice. In the upper left image the RGB image using Kinect is captured. In the upper right image, the point cloud from the same scene is captured. In the lower right image the probability map of the scene is shown, and in the right the final environment model is created. Note, that because of the perspective projection, the box in the front is much shorter in the three other images, than it can be seen in the RGB image. In the model, brighter objects are closer to the camera than darker areas.

Figure 17 the procedure to construct an environment model is demonstrated from the different aspects. In the first screen the RGB image from the Kinect is captured. The image is showing a plain wall and couple of boxes in the scene. The second screen is a capture of the point cloud from Kinect. It shows how the tall box (green color) in the front is much closer to the camera than the other objects. On the second row, the final environment model and probability map can be seen. In the

probability map, all the areas are equally dark as there is no motion in the scene. In the environment model the box in the front can be seen brighter than other areas, because it is much closer to the camera. The wide gap behind the box is caused by perspective projection of the 3-D point cloud data and Kinect together. Using the stereo imaging system, the gap would be narrower or even non-existent depending on the base line of the stereo system.

The experiments related to the environment model are performed using the Kinect sensor. The goal of the experiment is to demonstrate visually that the model can be updated successfully using the incoming point cloud data. The goal is also to show that the probabilistic nature of the algorithm works in practice and changing areas are not updated into the model.

4.2.1 Results

The results in this section are obtained by visual inspection of the developed environment modelling algorithm and the graphical interface as demonstrated in Figure 18.

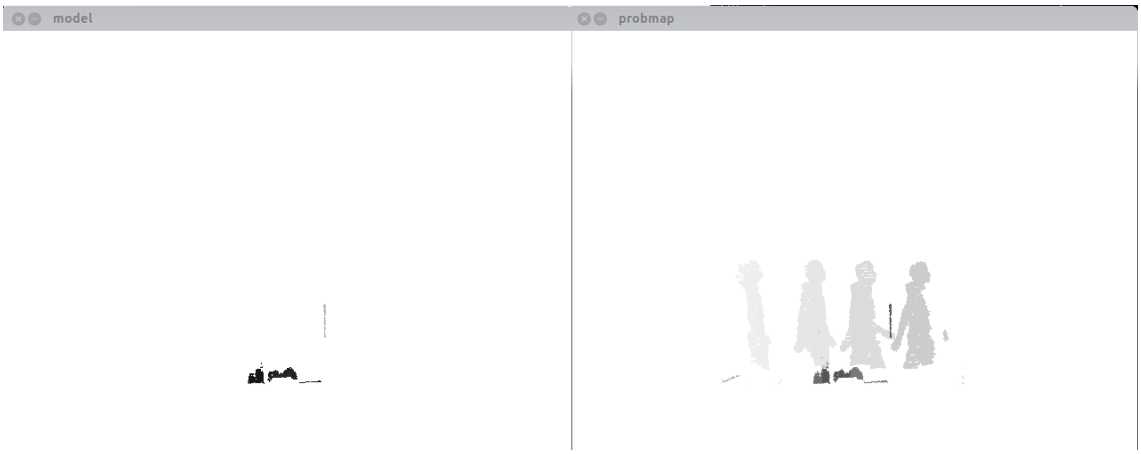

Figure 18: In the experiment a person is walking past the camera. On the environment model on the left, some static objects are visible. On the probability map on the right, static objects and the person walking past can be seen. The probability of occupied areas is decreased by every frame, which can be seen as illumination for the "old" and improbable areas. The colors of the image have been inverted to clarity on printed media.

In Figure 18 the updating procedure is visualized. In the image on the left, the static objects are visible. Those objects have been present during several frames so that the probability of those areas is high enough (the probability is visualized in the image on right) and they are added as static objects into the environment model. In this demonstration, the threshold for probability is 0.5, while having prior and

post probabilities 0.2 and 0.7 respectively. With the knowledge of the probability update formula Eq. (8), the minimum number of successive frames needed to get object into environment model can be calculated. Using a geometric sum

$$a + ar + ar^2 + ar^3 + ... + ar^{n-1} = \sum_{k=0}^{n-1} ar^k = a\frac{1-r^n}{1-r} \tag{22}$$

set $a = 0.2$ and $r = 0.7$, we obtain an equation

$$0.2\frac{1-0.7^n}{1-0.7} > 0.5 \tag{23}$$

By solving the equation, we get a result $n \geq 4$. With the chosen parameters, it means that, in order to get an object updated into the environment model, the object must be present in at least four successive images.

In Figure 19 another visualization of the performance of the algorithm is shown. There a person is standing in the front the the camera while holding a sheet of the paper in hand. On the second row of the images, the person is swinging his hand so that the movements from successive frames can be seen in probability map on the right hand side. The movement is fast enough so that the sheet or hand is not on the same position in two successive frames, only upper part of the arm is almost stationary so that area has been updated into environment model on the left. When the hand stops and stays still for four frames, it appears into model also (third row). Dim shadows of previous positions of the hand can still be seen in the probability map of the third row. Probability values of those areas are decreased towards the zero gradually.

The z-value of the object in the model is directly proportional to distance of the object in real environment. The more brighter the object in the model is, the closer it is to the camera. The region of study must be predefined before using the modeling algorithm. In Figure 19 x-axis (left-to-right) is bounded inside an area $[-3.2, 3.2]$, y-axis (up-to-down) is $[-3.2, 1.6]$, and z-axis (inside the image plane) $[-0.4, 1.6]$. Depth area is then 2 meters, and having 8-bit depth (255 values) in the model, the resolution of the depth is then $2/255 = 0.78$ cm. For x and y-axes the resolution is configured to be 0.01 meters/pixel, or 1 cm/pix.

Next, the model from third row of Figure 19 is investigated on a more specific level. The z value of the point in the center of the person's head is 175. The depth value in meters can simply be obtained by an equation $0.78 \cdot 175 = 137.25$ cm, or 1.37 m. Now, remember that z_min value is -0.4 so, to get correct distance between camera and head, 40 cm have to be subtracted from the result. Correct

distance is then $1.37 - 0.4 = 0.97$ m, or almost one meter. Another thing that can be measured from the model is, for instance, the width of the person's head. Using a measurement tool from any image processing application, 18 pixels as a maximum width of the person's head can be measured. Having horizontal and vertical resolution of 0.01 meters/pixel, the width of the head can be easily computed to be $0.01 \cdot 18 = 18$ cm, that is quite exactly the correct value.

During the testing period, input point clouds with a size of 768×576, equal to 442368 points, were used. First, cloud was downsampled to given resolution. If resolution is 0.01 and the area of the scene along the x-axis is 20 meters and along y-axis 11.5 meters, which are default values, the size of the model will be 2000×1150 meters respectively. Using the default resolution value 0.04 (1 pixel is 4 cm), size of the model will be 500×287, or 143500 cells. The resolution of the model influences updating time significantly. The durations for updating models with different resolutions are visualized in Figure 20. It shows that duration shortens when resolution halves.

4.2.2 Discussion

In the beginning of the project, either Octomap or pure point cloud seemed the most reasonable choice as an environment representation format. After noticing that input point cloud data might sometimes be quite noisy, it contains artifacts, and dynamic objects might be hard to detect, it was clear that some kind of history for the representation format would be necessary. Octomap contains probabilistic updating method that could have been suitable for this application, but during the initial tests it proved that updating the whole octree is computationally far too expensive operation using any reasonable resolution for data. The novel approaches like OM-NDT (*Occupancy Map – Normal Distribution Transformation*) would have take some time to implement, and it was not even clear that collision detection using OM-NDT could have been suitable for application. Decision to create entirely new presentation method was risky, but since the application was rather explicit and the first tests with the idea were promising, creating a new probabilistic environment model seemed reasonable.

The environment modeling algorithm presented in this thesis is a height map type, meaning that it is able to present one side of the scene successfully, but fails with overhanging objects like bridges. Another situation where it fails occurs when modeling a long object, such as a wall, looking it from the end of it. Only the end of the wall is visible on the model, and it contains no information about length of

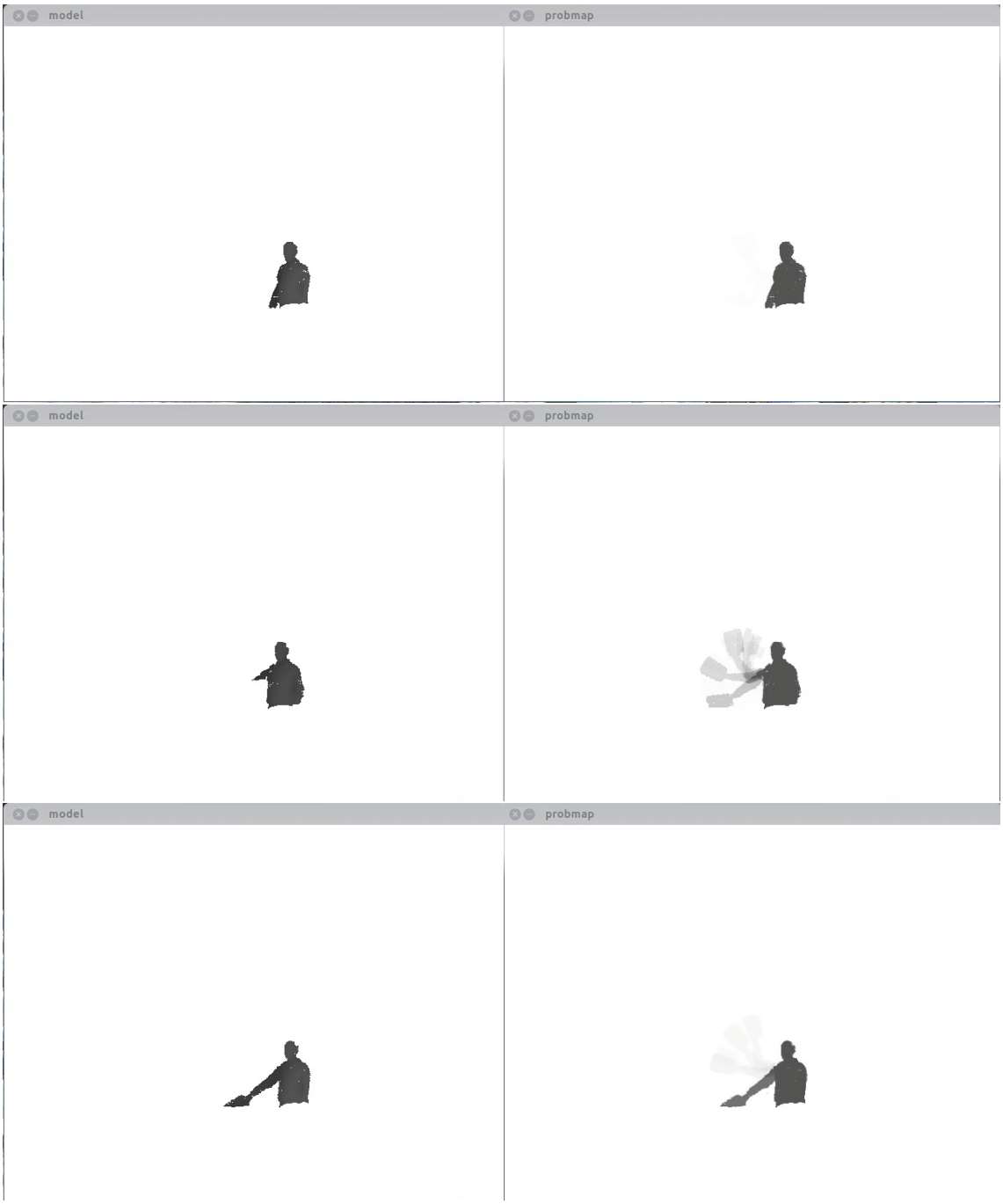

Figure 19: The screen captures of the environment modeling application are visualizing a person standing in front of the camera and swinging a sheet of the paper. On the first row, the window on the left is showing the current environment model generated based on a probability map on the right side. The images are similar because the person is standing still. On the second row the person is swinging the sheet, which is not visible on the environment model. On the third row the person stops swinging and the hand and the sheet appear back onto environment model on the left. The colors of the images are inverted to have a printed media more clarified. For the reference, the width of the person's head is approximately 18 centimetres.

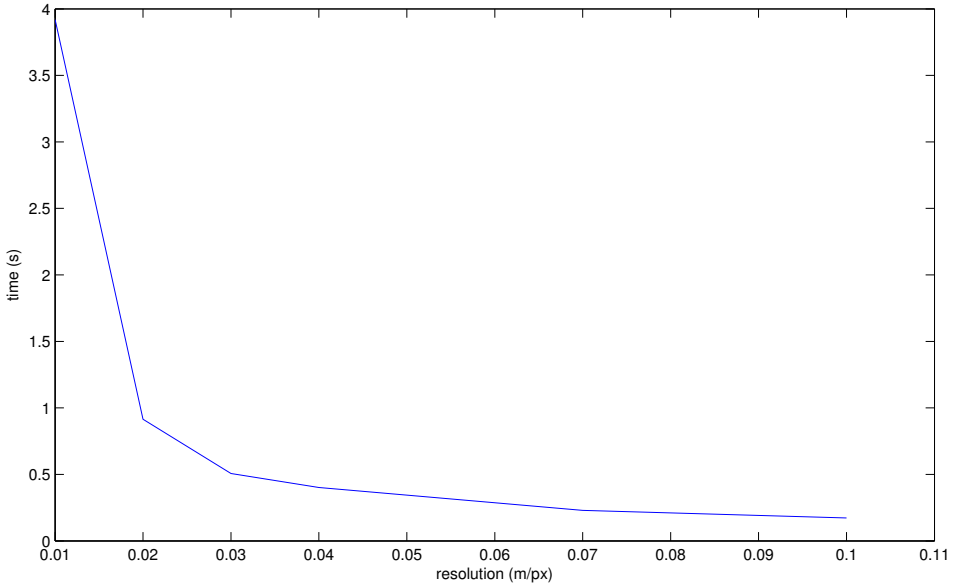

Figure 20: Duration of model updating compared using different resolution values.

the wall.

The immediate future work on this model could include developing of the probability update formula, as it is currently very plain, although workable. In addition, the prior and the post values of the updating algorithm are predefined to be $P_{prior} = 0.2$ and $P_{post} = 0.7$, but they are neither optimized anyhow nor studied if they are good at all. The totality is working, so those parameters must be good enough.

Time consumption values presented in Figure 20 are approximate and exceedingly related to used hardware and also to implementation of the algorithm. The implementation used during these tests is quite unsophisticated and unoptimized. The main bottle-neck is probably the loop where every 3-D point is reprojected on the 2D plane of the environment model one-by-one. Updating the model and other model-related operations are done efficiently on the image plane using optimized algorithms from OpenCV. Using reasonable large resolution value and limited size environment area, the updating times are satisfactory for this application.

4.3 Collision detection

Collision detection in 3-D environment as a whole is a multiphase and non-trivial task, but there exist a few ready-made libraries and algorithms for that. In [1] ODE

(*Open Dynamics Engine*) library was used to detect collisions between environment and a single object. Supported environment representations were a Pointcloud or Octomap, and the object could be in any geometric shape. If Pointcloud or Octomap was used in this project, as planned, ODE would have been the choice, but now when the custom environment representation was used, it was reasonable to use also custom-made collision checking algorithm.

Collision detection algorithm was presented in details in previous section. In this section the results of executing the implemented algorithm are presented. Unfortunately the final testing session in the real environment failed due the failures when producing input 3-D data from the stereo system, so the measurements from the real environment are non-existent. Some recorded offline data from the real environment were present, but the result validation procedure was unable to be performed. Simulated data was used instead.

To test the algorithm throughly, two testing applications were developed (see Fig. 21). The first application generates a pointcloud representation of a ground plane and a couple of cylinder shaped objects randomly to stand on the ground. The application then constructs a new environment model representation from the pointcloud. The environment model is now in a format that is presented in the previous section. Collision detection is then performed between the created environment model and another cylinder shaped object that is rotated to horizontal orientation. The object is translated along each coordinate axis, one by one, and the application performs collision detection algorithm each time the object moves. The second application is similar, but instead of vertical cylinders, a single wall is generated on the ground.

4.3.1 Results

In Table 6 execution times with different configurations are compared. Times are measured using the same hardware as presented in the previous section (see Table 3). *Resolution* value is reconfigurable constant for environment model, and X and Y values are the size of the model in meters. Using, for instance, resolution value 0.03, the size of the model can be obtained to be 20/0.03 times 11.5/0.03, which is a grid with 667×383 cells. *Interval* value is a distance between each search ray around the bounding volume, and *radius* is a length of the search ray. As can be seen, the execution times are small compared to environment model updating times in Figure 20. Doubling a resolution approximately halves execution time. Increasing a search radius or decreasing a search interval affects the execution time slightly, but

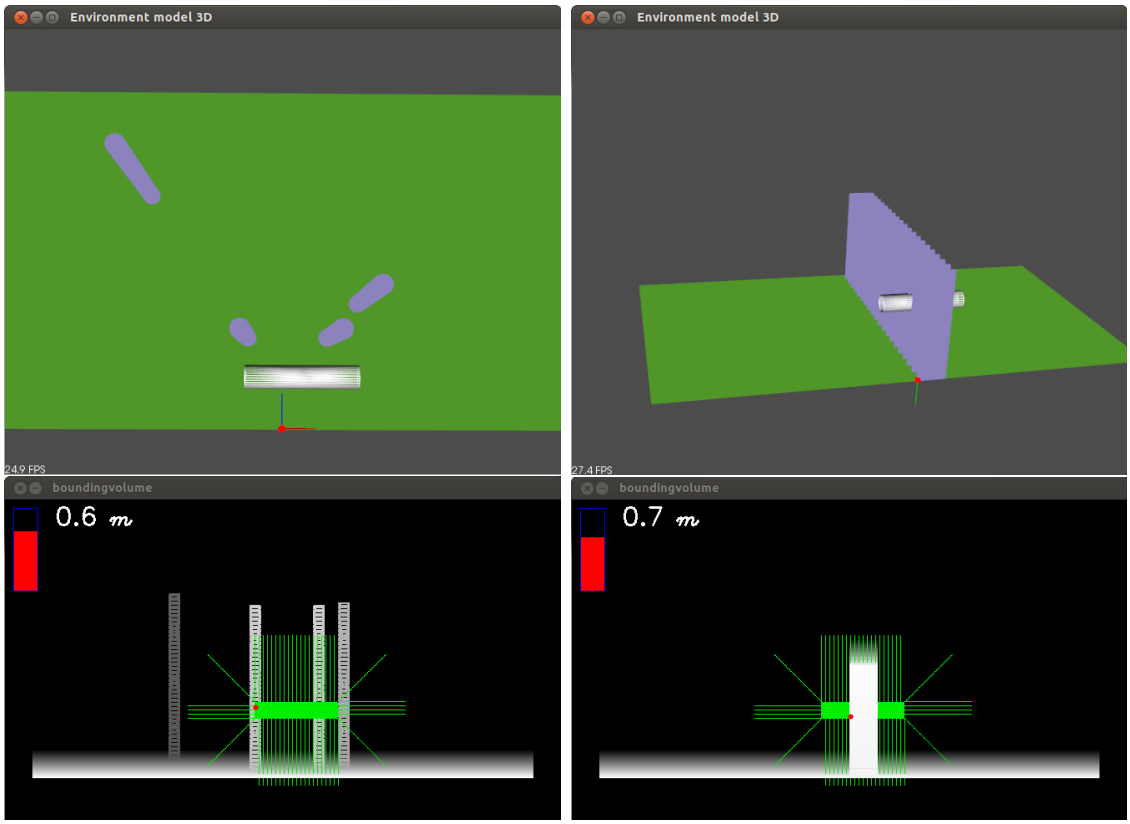

Figure 21: Two applications developed for testing collision detection algorithm. On the top-left the pointcloud representation of the ground floor and cylinders, and wire-representation of the colliding object viewed from the top. On the top-right the pointcloud representation of the ground, wall and colliding object. On the bottom row, environment model of the corresponding pointcloud, and green colored colliding object and search rays are shown. The result of collision detection algorithm can be seen as a distance to nearest object.

neither of them affects outstandingly. It is important to notice, that when resolution value halves (for instance from 0.02 to 0.01), the number of points along the search rays does not get increased to square, but it rather doubles.

To test the accuracy of the algorithm, two testing software were developed. Some screenshots of the developed applications are presented in Figure 22. They present three case, where four cylinder shaped objects are placed on the green floor plane. The cylinders are three meters high and have a diameter of 30 centimetres. Colliding object is a grey horizontal cylinder with a length of two meters and the same diameter as others. During the simulation, colliding object moves along the each coordinate axis, but in these screenshots they are currently moving along the z-axis from the front to the back. In the first case, the colliding object is in a front of all objects in the environment. As can be seen, there is clearly distance to the

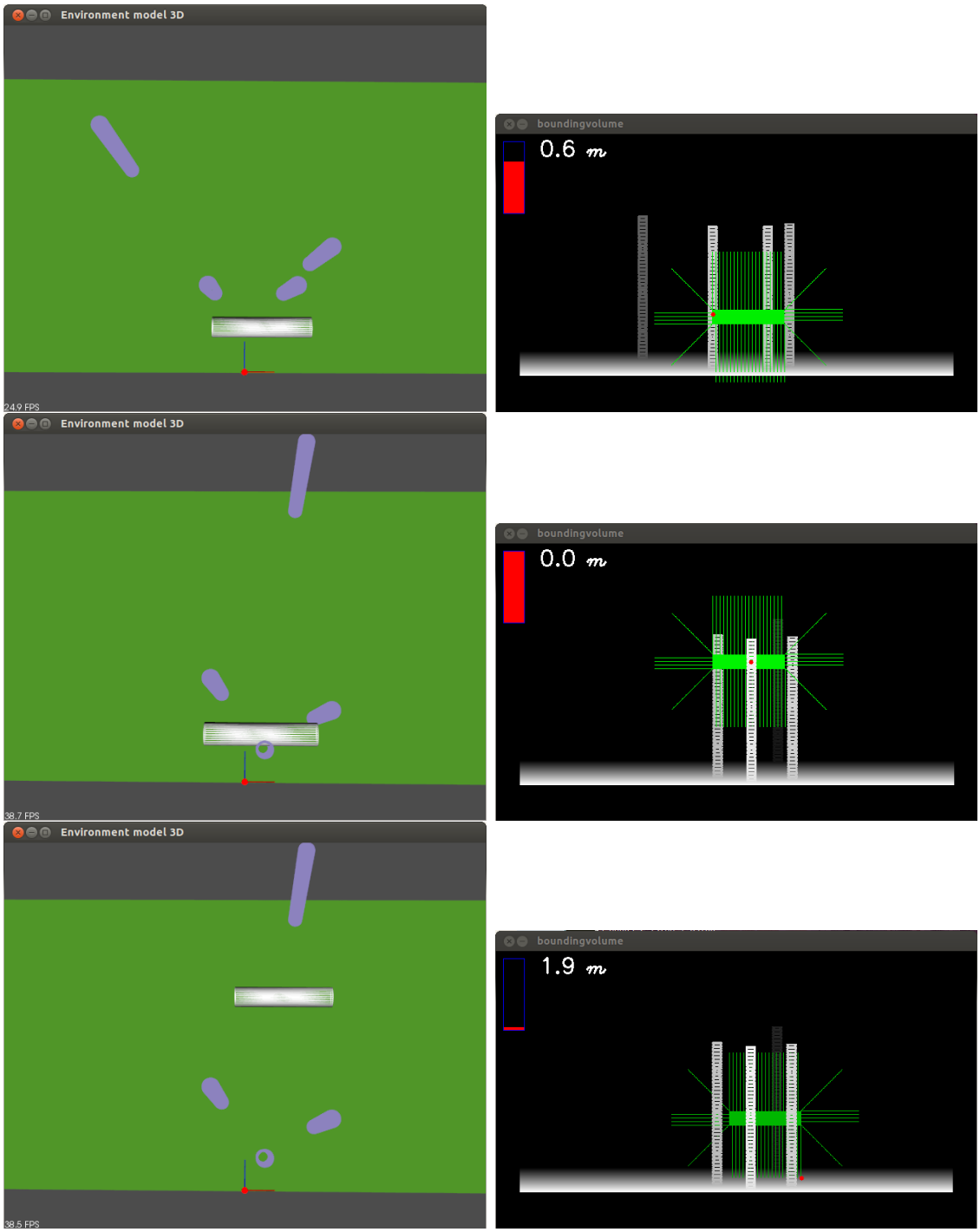

Figure 22: Screenshots of collision detection algorithm are visualizing the results of collision detection in three different cases. In all three cases (two images on the same row), four cylinders are placed on the ground (on the left: pointcloud viewed from the top, and on the right: environment model viewed from the front). Pointcloud representation is transformed to environment model and collision detection is performed using the environment model. In the first two cases the nearest point (collision point) is detected inside the bounding volume, and in the last case the collision point lies on the search ray and it is the point on the ground.

Table 6: Execution times of collision detection algorithm.

Resolution	X (m)	Y (m)	Interval (px)	Radius (px)	Time (ms)
0.01	20	11.5	8	70	20.46
0.02	20	11.5	8	70	9.97
0.03	20	11.5	8	70	7.12
0.05	20	11.5	8	70	4.33
0.02	20	11.5	8	100	12.70
0.02	20	11.5	8	70	9.97
0.02	20	11.5	5	70	8.32

nearest object. According the algorithm, the distance between the objects is 0.6 meters, which is the correct value. When the colliding object moves towards the objects, the distance decreases. On the second row, the case where collision has already occurred is shown. The colliding object is clearly inside the objects in the environment and distance to the nearest object is zero. On the third case, a distance to objects on the environment is large enough that the closest point can be found along the search rays. In this case the closest point lies on the ground floor and the distance between colliding object and that point is 1.9 meters.

In Figure 12 an example screenshot from the real offline dataset is presented. Input data is a pointcloud representing a machine lifting a cylindrical object (only a tip of the boom and an end-effector is visible). The cylinder and the end-effector are removed from the data (areas inside a red cube), and the pointcloud is then updated into probabilistic environment model in Figure 13. Then, collision detection algorithm is performed on that model in Figure 14. As can be seen, some artifacts (a pole in a front of the model) caused by stereo imaging are still visible in the model. However the distance between the cylinder and ground is calculated and the collision point is visualized with a red dot. In this case the distance of 3.7 meters is pretty much the correct value, albeit the real distances for this dataset are unavailable.

On the right in Figure 21 the second testing application developed is shown. The application is simple and the only purpose of it is to visualize the problems of overhanging or occluded objects with two-and-half dimensional algorithms like one developed during this thesis. As can be seen, the *wall* generated is parallel to the imaginary optical axis of the camera, and hence only a gable of the wall is visible on the projected model in the lower right corner. Because the collision detection is performed using that projection model, only the distance to gable can be measured. Although the colliding object is inside the wall, the collision detection algorithm is

claiming that nearest distance is 0.7 meters.

4.3.2 Discussion

While there exist multiple ready made libraries for collision detection and avoidance, even then, using the custom-made environment representation model, and having a rather simple application area, using a custom-made collision detection algorithm was reasonable. As demonstrated, the algorithm works well in a simple situation where the environment model is pure enough. As the environment model is a two-and-half dimensional, the overhanging or occluded objects might cause troubles to algorithm. Natural environment, such as forest, is an ideal environment for the algorithm, as the most of the objects there are simple bushes and trees, and long objects like walls are non-existent. The algorithm presented in this thesis is simple, fast, flexible, and easy to configure, in comparison to other available libraries such as ODE.

As mentioned before, the validation procedure of the results could not be performed as the final testing session was unsuccessful, so the results are mostly based on the simulator. In the real environment there are likely only a few objects, but they are not that well-shaped than in the simulator. Input data is not entirely pure and contains artifacts and noise. To compensate for these problems, the probabilistic environment model that filters out dynamic objects and small artifacts was developed. However, even with all those improvements compared to standard methods, some false objects or noise might appear into environment model. Using naive collision detection algorithms, the false collisions are probable to occur, when the algorithm suddenly detects the collision with a pixel sized object. As a future work, a probability measurement of oncoming or occurred collision could be to the purpose. Collisions with small objects would decrease the probability value where collisions, for instance, with ground floor would increase the probability measurement. Algorithm could also use the history knowledge to predict the direction and speed of colliding object and add weight to the side of bounding volume to the direction the object is moving.

5 Conclusion and future work

The goal of this thesis was to study methods to extend crane operator's spatial awareness when teleoperating the machine. Only a stereo imaging system was available, and it was able to produce one-sided range data of the environment containing end-effector and obstacles in a field of vision. Using the incoming range data, the adaptive environment model was built. The environment model developed during this thesis has nature of *two-and-a-half dimensional* depth image, since the incoming range data is received from a statically mounted stereo camera and only one side of the environment is visible. The method is able to extract dynamic parts of the environment due to its probabilistic nature. Using the developed environment model, also the collision detection system was built to warn operator for possibly oncoming collisions between the end-effector and the static environment. The distinct softwares involved in this thesis are environment modelling, collision detection, and target tracking applications. The latter of those is used to detect the 3-D coordinates of the end-effector of the crane from the image sequence, and then the boom and load is extracted from the model.

For target tracking, TLD is used in this project. TLD provides reliable and fast method to track the position of the pre-defined model. It is also capable to learn autonomously changes in the pose of the tracked object, and recover if the occlusion is occurred. Ready-made ROS implementation of the TLD exists, so implementation into the robot system is rather easy. During the tests, TLD proved to work commendably, it was properly suitable for this application, and there was no reason to improve it anyhow. Only issue to inspect as a future work would contain a construction of the single target model as starting point, when environment changes totally. At this moment, the target model is taught using multiple offline datasets, but when moving to a totally new environment, the model might not be suitable. It means that tracker is simply unable to detect the target, if for instance, illumination conditions changes. Also, the learning mode is switch off by default, because sometimes the tracker might "explode" by learning one false positive match, and using it, to start learning several false positive matches instead of marking them as negative matches. This problem occurs using this kind of autonomously learning trackers, and only way to avoid it might be to change the whole technique of the tracker to another one, for instance to CMT presented in Background section.

Environment modelling algorithm introduced in this thesis is unique method to represent 3-D information of the dynamic environment. The method is depth map type representation, which is extended with a probability layer indicating the

probability of each cell (or pixel) is occupied. The representation is possible to be used in any application with dynamic environment and static viewing point (camera). The advantage of the method is based on the probabilistic nature of the algorithm, which allows improbable areas to be filtered out. It means that all kind of artifacts or even some birds visible only in a few successive frames are excluded automatically. The disadvantage of the method is it stores only information visible to one viewing point, so that it is not able to store complete 3-D information of environment. Although it might be easily generalized to record a 360 degree field of view around the range sensor by using different kind of projection matrix, it still lacks the ability to store, for instance, depth of the objects visible in the environment model. In results section a wall viewed from the gable of the wall is given as an example.

Result section shows that developed algorithm is workable. The accuracy of the algorithm depends on the accuracy of the incoming 3-D data, as well as clarity of the model. If the data contains lots of artifacts or noise, also the environment model will be fuzzy, though most of those will be filtered successfully. Using the accurate range sensor such as Microsoft Kinect, the generated environment model is accurate enough to measure distances or even width of the humans head as presented. The algorithm is developed to be modular to apply it into different applications easily, it is also easily configurable to fit into different environments. The immediate future work on this model could include improving of the probability update formula, as it is currently very simple, although workable. The main bottle-neck in the algorithm is probably the loop where every 3-D point is reprojected on the 2D plane of the environment model one-by-one. It could be also improved a lot by optimizing the implementation.

Due to failures in setup during the final testing session in the real environment, the full action chain could not be tested, but distinct pieces of software were tested successfully using recorded offline data and developed simulators. The simulators were used especially to evaluate the functioning of the collision detection algorithm. The algorithm developed in this thesis is unique, mostly because it uses the environment model developed also in this thesis. In addition, the search rays the algorithm uses to detect obstacles nearby seems to be non-existent in the former literature in the area of collision detection. Mostly due to these features, the algorithm is multi-purpose and provides fast and reliable collision detection between the observed object and environment. In addition to ability to detect occurred collisions, algorithm is able to measure the distance to the nearest obstacle. As results

section presents, the algorithm works well in dynamic environment, if the incoming 3-D environment data is accurate enough. All kind of artifacts and noises encumber the accuracy and the functionality of the collision detection algorithm and might cause wrong alarms easily. Algorithm is developed especially for detecting collisions between a single object and the environment, but checking the collisions for multiple objects should be easily achieved.

As an immediate future work for the developed system, the experiments with real world data and the deep testing with a software module chain are required. After that, a probability measurement of the oncoming or the occurred collision for the collision detection algorithm would be to the purpose. Collisions with small objects would decrease the probability value where collisions, for instance, with ground floor would increase the probability measurement. Algorithm could also use the history knowledge to predict the direction and speed of colliding object and add weight to the side of bounding volume to the direction the object is moving. The algorithm would also be easy to generalize to detect the collisions in normal depth map or height map, as the probabilistic nature of the environment model is not essential for the collision detection algorithm.

Together the developed environment model and the collision detection algorithm could form a flexible, easily configurable, and probably working way to help an operator to observe the environment and avoid injurious collisions with obstacles when teleoperating the crane. However, at the moment the technique require still more accurate and cheaper range sensors, that it would be efficient and remunerative to use this kind of systems to help an operator to teleoperate a machine.

References

[1] A. Valli, "Sensor-based motion planning for a robotic manipulator," Master's thesis, School of Electrical Engineering, Aalto University, 2013.

[2] E. Mutanen, "Three-dimensional measurement of a lifted load using machine vision," Master's thesis, School of Electrical Engineering, Aalto University, 2014.

[3] B. Siciliano and O. Khatib, *Springer handbook of robotics*. Springer, 2008.

[4] H. P. Moravec and A. Elfes, "High resolution maps from wide angle sonar," in *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*, vol. 2, pp. 116–121, IEEE, 1985.

[5] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "Octomap: An efficient probabilistic 3d mapping framework based on octrees," *Autonomous Robots*, 2013. Software available at `http://octomap.github.com`.

[6] R. Triebel, P. Pfaff, and W. Burgard, "Multi-level surface maps for outdoor terrain mapping and loop closing," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pp. 2276–2282, IEEE, 2006.

[7] P. Biber and W. Straþer, "The normal distributions transform: A new approach to laser scan matching," in *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, vol. 3, pp. 2743–2748, IEEE, 2003.

[8] T. Stoyanov, M. Magnusson, H. Almqvist, and A. J. Lilienthal, "On the accuracy of the 3d normal distributions transform as a tool for spatial representation," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 4080–4085, IEEE, 2011.

[9] J. Saarinen, H. Andreasson, T. Stoyanov, J. Ala-Luhtala, and A. J. Lilienthal, "Normal distributions transform occupancy maps: application to large-scale online 3d mapping," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pp. 2233–2238, IEEE, 2013.

[10] F. W. DePiero and M. M. Trivedi, "3-d computer vision using structured light: Design, calibration, and implementation issues," *Advances in computers*, vol. 43, pp. 243–278, 1996.

[11] S. Foix, G. Alenya, and C. Torras, "Lock-in time-of-flight (tof) cameras: a survey," *Sensors Journal, IEEE*, vol. 11, no. 9, pp. 1917–1926, 2011.

[12] J. W. Weingarten, G. Gruener, and R. Siegwart, "A state-of-the-art 3d sensor for robot navigation," in *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, vol. 3, pp. 2155–2160, IEEE, 2004.

[13] R. Lange and P. Seitz, "Solid-state time-of-flight range camera," *Quantum Electronics, IEEE Journal of*, vol. 37, no. 3, pp. 390–397, 2001.

[14] MESA Imaging AG, "Swissranger sr4000." `http://www.mesa-imaging.ch/` (Retrieved February 10, 2014).

[15] J. Poppinga, A. Birk, and K. Pathak, *A characterization of 3D sensors for response robots*, pp. 264–275. RoboCup 2009: Robot Soccer World Cup XIII, Springer, 2010.

[16] C. Ye and J. Borenstein, "Characterization of a 2d laser scanner for mobile robot obstacle negotiation," in *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*, vol. 3, pp. 2512–2518, 2002.

[17] Velodyne Lidar Inc., "Hdl-64e." `http://velodynelidar.com/` (Retrieved February 10, 2014).

[18] T. Stoyanov, R. Mojtahedzadeh, H. Andreasson, and A. J. Lilienthal, "Comparative evaluation of range sensor accuracy for indoor mobile robotics and automated logistics applications," *Robotics and Autonomous Systems*, vol. 61, pp. 1094–1105, 10 2013.

[19] M. Hebert, "Active and passive range sensing for robotics," in *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, vol. 1, pp. 102–110 vol.1, 2000.

[20] C.-S. Chen, Y.-P. Hung, C.-C. Chiang, and J.-L. Wu, "Range data acquisition using color structured lighting and stereo vision," *Image and Vision Computing*, vol. 15, pp. 445–456, 6 1997.

[21] M. A. Livingston, *Vision-based tracking with dynamic structured light for video see-through augmented reality.* PhD thesis, The University of North Carolina at Chapel Hill, 1998.

[22] J. Salvi, J. Pages, and J. Batlle, "Pattern codification strategies in structured light systems," *Pattern Recognition*, vol. 37, no. 4, pp. 827–849, 2004.

[23] C. Guan, L. Hassebrook, and D. Lau, "Composite structured light pattern for three-dimensional video," *Optics Express*, vol. 11, no. 5, pp. 406–417, 2003.

[24] V. G. Yalla and L. G. Hassebrook, "Very high resolution 3d surface scanning using multi-frequency phase measuring profilometry," in *Defense and Security*, pp. 44–53, International Society for Optics and Photonics, 2005.

[25] Microsoft Corporation, "Kinect for xbox." `http://www.xbox.com/en-us/kinect/` (Retrieved February 25, 2014).

[26] K. Khoshelham, "Accuracy analysis of kinect depth data," in *ISPRS workshop laser scanning*, vol. 38, p. W12, 2011.

[27] P. Corke, *Robotics, vision and control: fundamental algorithms in MATLAB*, vol. 73. Springer, 2011.

[28] M. Okutomi and T. Kanade, "A multiple-baseline stereo," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 15, no. 4, pp. 353–363, 1993.

[29] E. Trucco and A. Verri, *Introductory techniques for 3-D computer vision*, vol. 201. Prentice Hall Englewood Cliffs, 1998.

[30] S. M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski, "A comparison and evaluation of multi-view stereo reconstruction algorithms," in *Computer vision and pattern recognition, 2006 IEEE Computer Society Conference on*, vol. 1, pp. 519–528, IEEE, 2006.

[31] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *International journal of computer vision*, vol. 47, no. 1-3, pp. 7–42, 2002.

[32] G. Slabaugh, B. Culbertson, T. Malzbender, and R. Schafer, "A survey of methods for volumetric scene reconstruction from photographs," in *Proceedings of the 2001 Eurographics conference on Volume Graphics*, pp. 81–101, Eurographics Association, 2001.

[33] A. D. Luca, A. Albu-Schaffer, S. Haddadin, and G. Hirzinger, "Collision detection and safe reaction with the dlr-iii lightweight manipulator arm," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pp. 1623–1630, IEEE, 2006.

[34] S. Morinaga and K. Kosuge, "Collision detection system for manipulator based on adaptive impedance control law," in *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, vol. 1, pp. 1080–1085, IEEE, 2003.

[35] A. Chakravarthy and D. Ghose, "Obstacle avoidance in a dynamic environment: A collision cone approach," *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, vol. 28, no. 5, pp. 562–574, 1998.

[36] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[37] M. Lin and S. Gottschalk, "Collision detection between geometric models: A survey," in *Proc. of IMA conference on mathematics of surfaces*, vol. 1, pp. 602–608, 1998.

[38] J. Klein and G. Zachmann, "Point cloud collision detection," in *Computer Graphics Forum*, vol. 23, pp. 567–576, Wiley Online Library, 2004.

[39] P. Renton, M. Greenspan, H. A. Elmaraghy, and H. Zghal, "Plan-n-scan: A robotic system for collision-free autonomous exploration and workspace mapping," *Journal of Intelligent and Robotic Systems*, vol. 24, no. 3, pp. 207–234, 1999.

[40] C. A. Shaffer and G. M. Herb, "A real-time robot arm collision avoidance system," *Robotics and Automation, IEEE Transactions on*, vol. 8, no. 2, pp. 149–160, 1992.

[41] N. Burtnyk and M. A. Greenspan, "Real time collision detection," 1994.

[42] P. M. Hubbard, "Interactive collision detection," in *Virtual Reality, 1993. Proceedings., IEEE 1993 Symposium on Research Frontiers in*, pp. 24–31, IEEE, 1993.

[43] S. Kockara, T. Halic, K. Iqbal, C. Bayrak, and R. Rowe, "Collision detection: A survey," in *Systems, Man and Cybernetics, 2007. ISIC. IEEE International Conference on*, pp. 4046–4051, 2007.

[44] M. C. Lin and D. Manocha, "Efficient contact determination between geometric models," 1994.

[45] B. V. Mirtich, "Impulse-based dynamic simulation of rigid body systems," 1996.

[46] B. Mirtich, "Efficient algorithms for two-phase collision detection," *Practical motion planning in robotics: current approaches and future directions*, pp. 203–223, 1997.

[47] M. C. Lin and J. F. Canny, "A fast algorithm for incremental distance calculation," in *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pp. 1008–1014, IEEE, 1991.

[48] M. Moore and J. Wilhelms, "Collision detection and response for computer animation," *ACM Siggraph Computer Graphics*, vol. 22, no. 4, pp. 289–298, 1988.

[49] B. Mirtich, "V-clip: Fast and robust polyhedral collision detection," *ACM Transactions on Graphics (TOG)*, vol. 17, no. 3, pp. 177–208, 1998.

[50] S. A. Ehmann and M. C. Lin, "Accelerated proximity queries between convex polyhedra by multi-level voronoi marching," in *Intelligent Robots and Systems, 2000.(IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on*, vol. 3, pp. 2101–2106, IEEE, 2000.

[51] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi, "A fast procedure for computing the distance between complex objects in three-dimensional space," *Robotics and Automation, IEEE Journal of*, vol. 4, no. 2, pp. 193–203, 1988.

[52] D. Knott, "Collision and interference detection in real time using graphics hardware," 2003.

[53] N. K. Govindaraju, S. Redon, M. C. Lin, and D. Manocha, "Cullide: Interactive collision detection between complex models in large environments using graphics hardware," in *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pp. 25–32, Eurographics Association, 2003.

[54] E. Guendelman, R. Bridson, and R. Fedkiw, "Nonconvex rigid bodies with stacking," in *ACM Transactions on Graphics (TOG)*, vol. 22, pp. 871–878, ACM, 2003.

[55] P. Jiménez, F. Thomas, and C. Torras, "3d collision detection: a survey," *Computers & Graphics*, vol. 25, no. 2, pp. 269–285, 2001.

[56] S. Kimmerle and C. R.-D. Objects, "Bounding volume hierarchies," 2005.

[57] J. T. Klosowski, M. Held, J. S. Mitchell, H. Sowizral, and K. Zikan, "Efficient collision detection using bounding volume hierarchies of k-dops," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 4, no. 1, pp. 21–36, 1998.

[58] D. H. Ballard, "Strip trees: a hierarchical representation for curves," *Communications of the ACM*, vol. 24, no. 5, pp. 310–321, 1981.

[59] G. Zachmann, "Exact and fast collision detection," 1994.

[60] S. Gottschalk, M. C. Lin, and D. Manocha, "Obbtree: A hierarchical structure for rapid interference detection," in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pp. 171–180, ACM, 1996.

[61] B. K. Horn and B. G. Schunck, "Determining optical flow," in *1981 Technical Symposium East*, pp. 319–331, International Society for Optics and Photonics, 1981.

[62] J. J. Gibson, *The perception of the visual world.* Houghton Mifflin, 1950.

[63] J. L. Barron, D. J. Fleet, and S. S. Beauchemin, "Performance of optical flow techniques," *International journal of computer vision*, vol. 12, no. 1, pp. 43–77, 1994.

[64] B. D. Lucas, T. Kanade, *et al.*, "An iterative image registration technique with an application to stereo vision.," in *IJCAI*, vol. 81, pp. 674–679, 1981.

[65] C. Tomasi and T. Kanade, *Detection and tracking of point features.* School of Computer Science, Carnegie Mellon Univ. Pittsburgh, 1991.

[66] J. Shi and C. Tomasi, "Good features to track," in *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on*, pp. 593–600, IEEE, 1994.

[67] Z. Kalal, K. Mikolajczyk, and J. Matas, "Tracking-learning-detection," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 34, no. 7, pp. 1409–1422, 2012.

[68] G. Nebehay, "Robust object tracking based on tracking-learning-detection," *Vienna: Faculty of Informatics, Vienna University of Technology*, 2012. Software is reachable in `http://www.gnebehay.com/tld/` (Retrieved on March 26, 2014).

[69] R. Chauvin, "Opentld for ros." `https://github.com/Ronan0912/ros_opentld` (Retrieved on March 26, 2014).

[70] G. Nebehay and R. Pflugfelder, "Consensus-based matching and tracking of keypoints for object tracking," in *Winter Conference on Applications of Computer Vision*, IEEE, Mar. 2014.

[71] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, 2009.

[72] W. Garage, "Robot operating system." `http://www.ros.org` (Retrieved March 26, 2014).

[73] Itseez, "Open source computer vision library - opencv." `http://opencv.org/` (Retrieved on March 26, 2014).

[74] R. B. Rusu and S. Cousins, "3d is here: Point cloud library (pcl)," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 1–4, IEEE, 2011.